Carnegie Mellon
Software Engineering Institute

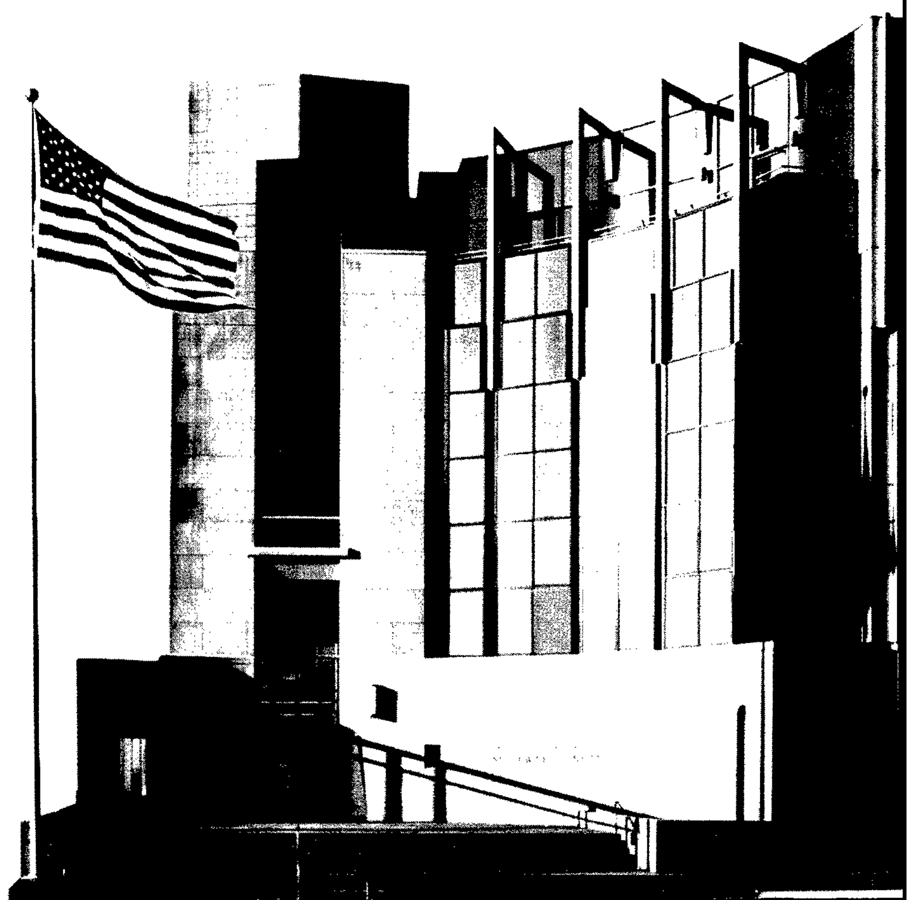
Interactions Among Techniques Addressing Quality Attributes

Hernan R. Eguiluz, Carnegie Mellon School of Computer Science

Mario R. Barbacci, Software Engineering Institute

June 2003

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited



TECHNICAL REPORT
CMU/SEI-2003-TR-003
ESC-TR-2003-003

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Interactions Among Techniques Addressing Quality Attributes

CMU/SEI-2003-TR-003
ESC-TR-2003-003

Hernan R. Eguiluz, Carnegie Mellon School of Computer Science

Mario R. Barbacci, Software Engineering Institute

June 2003

Architecture Tradeoff Analysis Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

There is very little published work on how techniques that promote different architectural qualities interact with each other. When developing a software system, software architects need to understand the relationships among these techniques. For example, if a system is compromised, architects must consider questions such as whether it makes sense to apply damage confinement to achieve dependability, while at the same time shutting down components to promote security. To help answer such questions, this report provides matrices in which various techniques for promoting different architectural qualities are analyzed relative to each other. Four architectural qualities were analyzed: performance, security, modifiability, and dependability. The techniques that promote each one were selected and categorized as promotion, detection, or correction. For each category, matrices are presented that provide a detailed description of why a particular interaction is positive, negative, or neutral, or cannot be determined without assessing a concrete system.

1 Introduction

This report was conceived from the realization that there is very little published work on how techniques that promote different architectural qualities interact with each other. For example, if the system is compromised, does it make sense to apply damage confinement to achieve dependability, while at the same time shutting down components? This and many other similar questions are considered by an architect when developing a software system.

This report is an attempt to provide software architects with a chart for determining the relationships among techniques that promote different architectural qualities. In addition, we hope that this report will help to bring awareness of the relationships among techniques to the communities that specialize in architectural qualities. More communication across these communities is needed.

The four architectural qualities that were selected for this report are defined below:

1. **performance:** the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage [IEEE 90]
2. **security:** the subfield of information science concerned with ensuring that information systems are imbued with the condition of being secure, as well as the means of establishing, testing, auditing, and otherwise maintaining that condition [Allen 99]
3. **modifiability**: for software products, the extent to which the product facilitates the incorporation of changes, once the nature of the desired change has been determined [Boehm 78]; for a software system, the ease with which the system can be modified to changes in the environment, requirements, or functional specification [Lassing 02]
4. **dependability:** the ability to deliver service that can justifiably be trusted. The service delivered by a system is its behavior, as perceived by its user(s); a user is another system (physical or human) that interacts with the former system at the service interface [Laprie 92]

The techniques that promote each of these qualities were selected and categorized into three groups:

1. **promotion:** This group includes those techniques that are used to achieve a given architectural quality attribute.
2. **detection:** This group includes techniques that are used to detect deviations from achieving the desired quality attribute once a system has been deployed.

3. **correction:** In those cases where the detection techniques find a deviation, this group of techniques is used to return the quality attribute to its desired value or reinstate it.

For each of these groups, we created a matrix in which each technique is analyzed relative to each of the other techniques in the same group. The relationship between pairs of techniques is expressed in terms of the following symbols (shown in Table 1):

Table 1: Key for Matrix Symbols

	The two techniques collide, and an architect may find it very difficult to support the two techniques in the same architecture.
	The two techniques work very well with each other; they may even facilitate each other. In this case, an architect will be encouraged to use both techniques together.
=	The two techniques are independent of each other. They can coexist in the same architecture without disturbing or helping each other.
?	The type of interaction between the two techniques (e.g., positive or negative) depends on the system being studied. The result of the interaction cannot be generalized.

Grey rows correspond to interactions that were not analyzed because we assumed that the interactions were symmetric.

1.1 Limitations

Comparing every technique that would promote the qualities selected would have been impossible. Therefore, for this report, we concentrated on those techniques that are widely used by practitioners. We didn't include techniques proposed by researchers that are either experimental or that have not been widely accepted by industry. Still, given the scope of this work, we concentrated on breadth, covering many techniques instead of selecting a few and then performing an in-depth analysis. Such in-depth analysis is left for future research.

In addition, to simplify our analysis, we assumed that interactions are symmetric. Symmetry means that the interaction between techniques A and B is the same as that between B and A. For most cases, this is valid.

1.2 Intended Audience

This report was written with practicing software architects in mind. It assumes that the reader has some basic knowledge of software architecture and understands the concept of quality attributes.

1.3 Outline of This Report

In Section 2, we present our basic understanding of an interaction between techniques. In Section 3, the definitions for all the techniques are presented. Section 4 presents the results of the interaction among all the techniques in the form of matrices: one for each group of techniques. Finally, in the appendices, the matrices that detail every row of the summary matrix are presented. These detailed matrices provide all the information that readers need to understand why we believe that an interaction is positive, negative, neutral, or undetermined.

2 The Idea of Interacting Techniques

The fundamental theme of this report is the study of the interactions between different techniques that are used to promote architectural qualities in software systems. For this report to be effective, we need to define what we mean by the term *interaction*. Following is a series of examples to demonstrate our idea of interacting techniques and why it is important.

The key shown in Figure 1 will be used for all the examples in Chapter 2. It won't appear next to each diagram to improve the flow of the explanation.

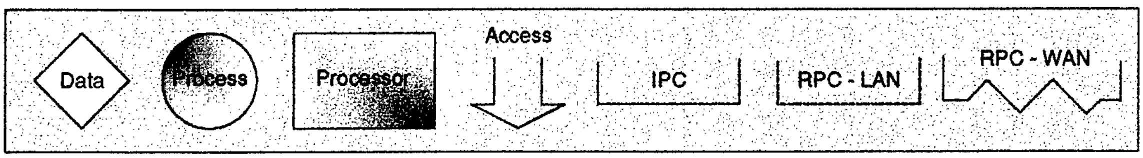


Figure 1: Key for Examples

Techniques appear in italics (e.g., *separation of concerns*) to highlight them within the text.

Sections 2.1 and 2.2 exemplify the kinds of techniques that can be applied successively after applying techniques to promote dependability and modifiability, respectively. It was not our intention to cover every single possibility in each case but to present valid and realistic alternatives that a software architect may consider. Furthermore, the examples use an abstraction of a system for reasons of brevity and, more importantly, to focus the reader on the architectural qualities and the techniques used to achieve those qualities.

2.1 Promoting Dependability

We will begin with a simple system, as shown in Figure 2. It consists of a single process located on a single processor. This process (that could represent a system) has a single access point and data repository.

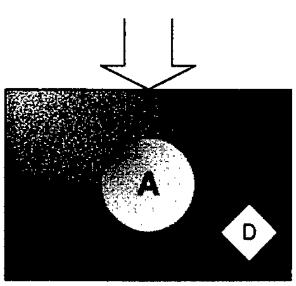


Figure 2: Single-Process System

This system doesn't promote dependability because any failure will mean that the system will stop providing its services. Security could be achieved if the system is properly configured. Modifiability could or could not be present; this view is too coarse grained to tell. Finally, performance may be adequate for a single user but probably not for multiple users and high volumes of data.

Concentrating on dependability, process A could be *replicated*, creating the system shown in Figure 3.

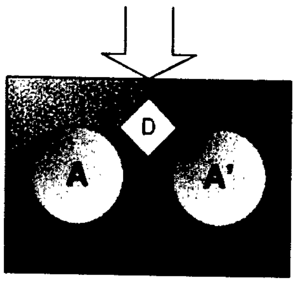


Figure 3: Two-Process System with Shared Data

Now the system supports a software failure, and its copy (A') will take over processing. Hence, this new system has better dependability. Security, on the other hand, may or may not have degraded. It is possible that a user will now have access to both A and A'. If this is the case, both need to be secured accordingly to guarantee data consistency. One option would be the use of *cryptography* that, if added to A, will automatically be part of A' too. This is a better option than adding *access control* from the point of view that A and A' may need different configurations. Yet, by having A and A', it is easier to configure the system to be more survivable to an attack.

From a performance standpoint, if the original system was using *replicated data*, this may or may not carry over to the new configurations. One of the most common ways to achieve dependability through replication is by eliminating all state from the servers. Then, storing partial results in *replicated data* will probably no longer be an option. If the system is part of a larger real-time system, rate monotonic analysis *(RMA)* techniques could have been used to establish its schedulability. However, this technique breaks when used in the presence of a system that could fail. Real-time dependable systems are currently an active research subject, and there is no definite solution to the problem [Natarajan 00, Powell 88].

Even though the second system (Figure 3) is more dependable than the first system (Figure 2), a hardware problem will take both replicas out of service. The usual solution to this problem is to put the replicas on different physical systems. This renders a third system, shown in Figure 4.

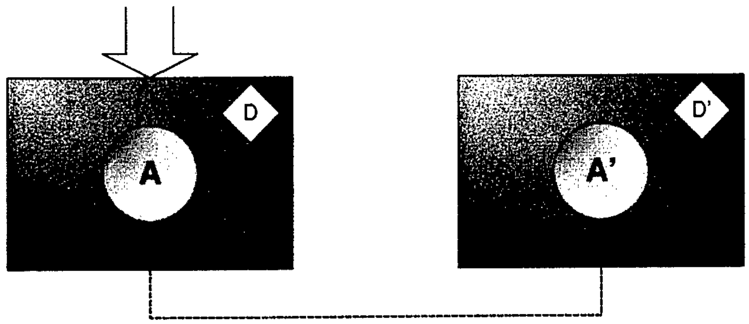


Figure 4: Replicated Processes on Separate Processors

In this third system, one of the processors can be taken out of service and the second one will still be capable of providing services. Although the dependability problem has been solved to a certain degree,[1] a security problem has now emerged; the information that moves between the two processes is no longer secure. If an industry standard protocol like Transmission Control Protocol/Internet Protocol (TCP/IP) is being used, the information on the wire can be snooped and altered. To prevent this, *cryptography* techniques are usually used. Using these techniques adds costs in terms of processing power dedicated to encryption and decryption on both processors.[2] In addition, if active replication is used, A and A' must finish executing an operation before a new one can be executed. This requires A and A' to synchronize themselves, which makes the overall system much slower due to the presence of a network connection. Then, *distribution* is no longer an option to increase performance. Otherwise, the synchronization penalty associated with distributed processing will most likely offset any benefit gained.

If the two replicas are connected by a wide area network (WAN), as shown in Figure 5, both the performance and security problems are exacerbated. *Distribution* must be ruled out completely is this case. Because there is a second physical processor running in a separate location, *access control* is not only required but difficult because the main purpose of the second processor may no longer be A'. Access control may need a compromise between the needs of A' and some other process B. Furthermore, the administrators assigned to the configuration of the two processors are probably different, adding to the complexity of *access control* configuration.

1 The system, as shown, can support one point of failure.

2 Network Interface Cards (NICs) that take care of the cryptography could be used, reducing this problem.

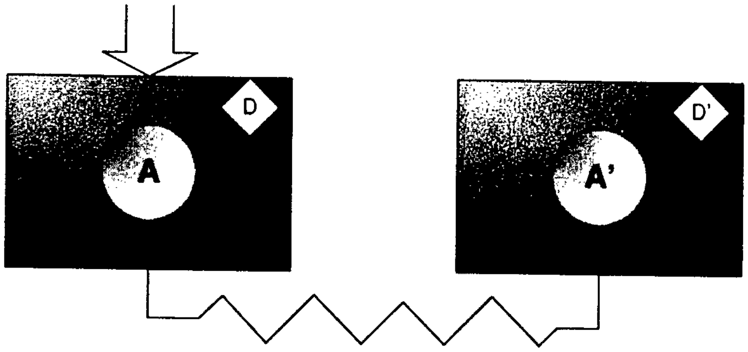


Figure 5: Replicated Processes Connected by a WAN

The most likely scenario in this case will be that both A and A' will be accessible to users to increase the system's responsiveness (performance). Then, the system shown in Figure 6 is achieved.

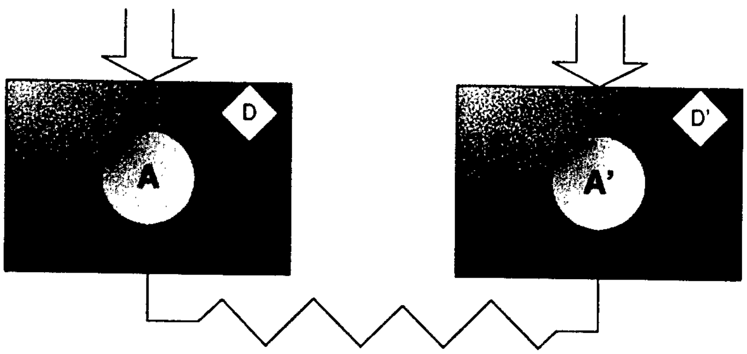


Figure 6: Replicated Processes with Different Access Points

Now, security problems arise because two identical copies of the system can be accessed from different access points. As an example, the system now needs to coordinate access control between A and A'.

As shown, when using a technique to promote an architectural quality, some qualities are promoted, while others are reduced.

2.2 Promoting Modifiability

At this point, we want to explore a different evolution path for the system composed of Process A on a single processor. As a reminder, the initial system configuration (previously shown in Figure 2) is shown again in Figure 7.

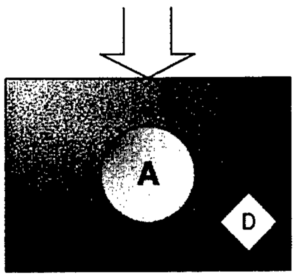


Figure 7: Single-Process System

In this case, let's assume that, due to changes that were too difficult in the original system, separation of concerns was applied to simplify A's maintenance by two different teams. The resulting system is shown in Figure 8.

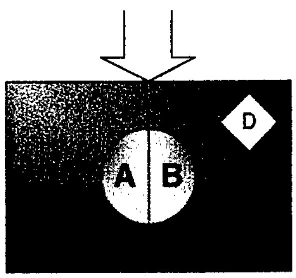


Figure 8: System with Separation of Concerns Applied

This division also simplifies running A and B as different processes (*concurrency*), which increases the perceived performance of the overall system. There may be some shared memory between the two processes, which must be encapsulated (*information hiding*) so that any one of the processes can access its own data (*data division*). By dividing the data (as shown in Figure 9), we are discouraging *Markov models* and *replicatio*n due to their added complexity in the presence of *data division.* Although dividing the data is a larger effort, which reduces performance slightly, in most cases, this reduction in performance is minor.

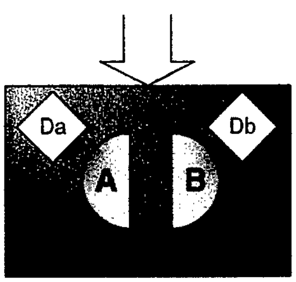


Figure 9: Separate Processes with Data Division

To increase performance, this can be taken one step further (as shown in Figure 10) by using a second processor to host B (*concurrency*).

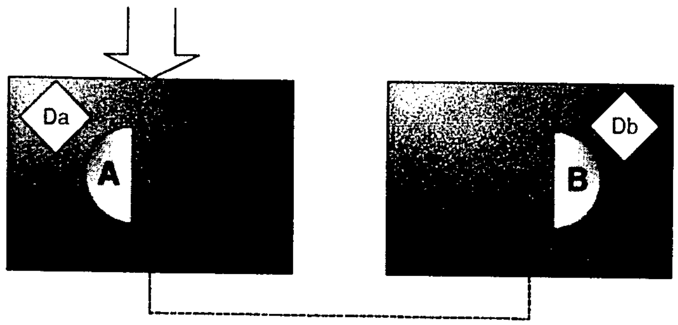


Figure 10: Separate Processes on Separate Processors

Although this architecture looks better than the system's previous incarnation (Figure 9), the following problems are introduced in this new architecture:

- The same security problems outlined for the replication case (Figure 4) are also valid here.
- Although there are no coordination problems due to replication, A and B still need to interact. When designing a system to be distributed, the interfaces between A and B would be minimized. Given that the original system was not conceived to reside on separate processors, there may be more coupling between A and B than strictly needed. This will affect performance.
- If shared memory were used to improve performance in the communication between A and B, a major problem would arise. Either the system would need to be redesigned in this respect, or the data would need to be replicated in A and B.

If the two processes are connected by a WAN (as shown in Figure 11), *replication* could be added to the system, increasing its complexity and testing effort in part because *performance engineering* would become more difficult.

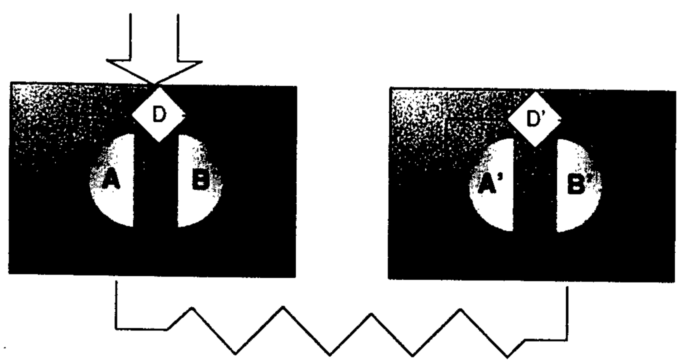


Figure 11: Replication of Separate Processes

This system returns coupling to its original level and shared memory is an option again, but performance is lost due to the processes being collocated. On the positive side, the system has gained dependability. But wouldn't it be better if processing could be distributed again (*concurrency*)? This new architecture (shown in Figure 12) would promote dependability and performance.

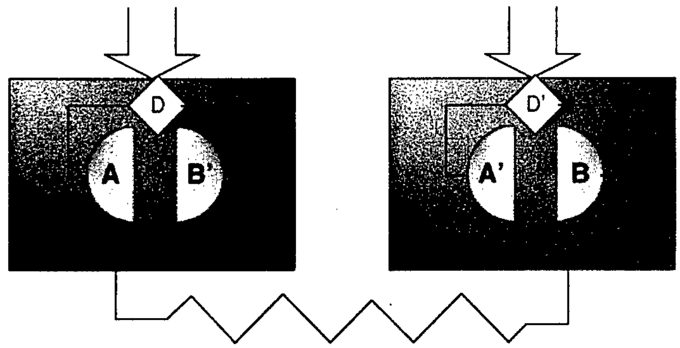


Figure 12: Distributed Processing

Indeed, this architecture allows for distributed processing, improving performance (or not, depending on the coupling between A and B). Dependability is improved, too. This architecture is even better than the previous architecture because if one of the processors is to be removed from service, only one replica must be promoted to primary. However, there are now two access points, making security a larger problem (*access control*).

This chapter has shown that different techniques that seem appropriate in isolation may not interact correctly when combined. Furthermore, applying one technique may prevent another one from being applied.

3 Techniques Used

In this chapter, we provide the definitions of the different techniques that were studied. The definitions are grouped into the following categories, which were defined in Chapter 1: promotion, detection, and correction. Within these groups, the techniques are further classified based on the quality attribute that they promote. Furthermore, the techniques appear in the same order as they do in the interaction matrices.

3.1 Definitions of Promotion Techniques

3.1.1 Security

1. **cryptography:** These techniques are used to achieve one or more of the following: confidentiality, authentication, integrity, and non-repudiation of information [Viega 02].
2. **access control:** This technique has two very distinct aspects. System access control involves ensuring that unauthorized users don't get into the system and encouraging (and sometimes forcing) authorized users to be security conscious. Data access control, on the other hand, monitors who can access what data and for what purpose. The system can determine access rules based on the security levels of the people, the files, and the other objects in your system [Russell 91].
3. **survivability**: This technique is used to analyze the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents. The system is used in the broadest possible sense, including networks and large-scale systems of systems. The mission is a set of very high-level requirements or goals. The terms *attack, failure*, and *accident* are meant to include all possible damaging events; but these terms do not partition these events into mutually exclusive or even distinguishable sets [Ellison 97].
4. **threat assessment:** This technique is used to determine what possible threats a system may face. The environment/context where the system will reside is a key source of threats because it determines what is and is not possible.
5. **vulnerability analysis:** This set of techniques is used to find vulnerable points in the software and hardware components of a system. These points are based on threat assessment and other information like the programming language or languages in which the system will be written [Krsul 98a, Krsul 98b].

3.1.2 Performance

6. **rate monotonic analysis:** This technique includes a collection of quantitative methods and algorithms that allow engineers to specify, understand, analyze, and predict the timing behavior of real-time software systems [Klein 93].

7. **performance engineering:** This technique is defined as "... a systematic, quantitative approach to constructing software systems that meet performance objectives. It uses model prediction to evaluate tradeoffs in software functions, hardware size, quality of results, and resource requirements" [Smith 02].

8. **data replication:** This technique uses local copies of information stored in a component that enables them to be accessed more quickly than from their original location.

9. **process replication:** This technique executes the same process on multiple instances of a hardware platform. Performance can be improved by using the aggregate computing power of all the replica sites on a single load category [Helal 96].

10. **data division:** This technique consists of splitting the data used by different subsystems into sets that have a property (like allowing parallel access by parallel processes) that is beneficial to the overall system performance.

11. **process division:** This technique consists of splitting a task between processes that work in parallel to reduce the overall time to complete the task.

3.1.3 Dependability

12. **testing:** This technique is the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [IEEE 90].

13. **Markov modeling:** This technique uses Markov chains for dependability prediction for fault-tolerant systems. It can model much of the combinatorial and sequence-dependent behavior that other models do in addition to using complex repair strategies, dynamic reconfiguration using spares, and complex fault/error recovery procedures that are not always perfectly effective [Boyd 96].

14. **replication:** These techniques are used to implement the two fault-tolerance activities of masking failures and reconfiguring the system in response to a failure [Helal 96].

3.1.4 Modifiability

15. **change scenarios:** In this technique, sequences of events that will change an architecture are created. These sequences are then used to assess their impact on the system. They are concrete, thus enabling detailed statements about their impact [Lassing 02].

16. **separation of concerns:** This technique is an approach to divide the inherent complexity of the software into more manageable units. In an ideal world, these concerns could be investigated separately and then integrated to create a whole solution [Savolainen 00].

17. **information hiding:** This is a software development technique in which each module's interfaces reveal as little as possible about the module's inner workings, and other modules are prevented from using information about the module that is not in the module's interface specification. (In summary, information hiding is a software development technique that consists of isolating a system function or a set of data and operations on those data within a module and providing precise specifications for the module [IEEE 90].)

3.2 Definitions of Detection Techniques

3.2.1 Security

1. **logging:** This technique consists of registering on permanent storage a set of activities that are relevant to the detecting security breaches. For logging to be effective, monitoring has to be put in place.

2. **monitoring:** This technique relies on logging to provide it with activities and events that are happening in the system. Its main objective is to scan those activities to find possible security breaches. For example, it can represent reviewing access logs and looking at packets moving on the network.

3. **honey pot:** This technique promotes the use of misinformation to throw off attackers and to facilitate the detection of malicious activities. This technique is valid for both internal and external attackers [Ellison 01].

3.2.2 Performance

4. **time-out:** This technique relies on the detection of processes that cannot respond to simple "heartbeat" queries because they are overloaded. In this case, the process that needs to respond to the heartbeat requests is busy, and, therefore, its performance might not be adequate.

5. **missed deadlines:** This technique relies on a real-time system's ability to detect that its processes are taking longer to finish than they should.

3.2.3 Dependability

6. **triple modular redundancy (TMR):** This technique is the evolution of Von Neumann's example of a redundancy scheme that is used for masking faults [Von Neumann 56]. In a TMR system, three implementations (which might be the same or different) of the same logic function are used, and the outputs of all the implementations are connected to a voter [Mitra 00].

7. **recovery blocks:** This technique, as described by D. Nguyen, ". . . consists of three software elements: (1) a primary module, which executes critical software functions; (2) an acceptance test, which tests the output of the primary module after each execution; and (3) at least one alternate module which performs the same function as the primary module (but may be less capable or slower) and is invoked by the acceptance test upon detection of a failure" [Nguyen 98].

3.2.4 Modifiability

8. **time assessment:** This technique relies on identifying an increasing time required to modify a system compared to previous similar modifications.
9. **defect assessment:** This technique relies on identifying an escalating number of defects introduced to a system regardless of the size of the proposed modification.
10. **impact assessment:** This technique relies on identifying a reduction of the impact in terms of the number of modules affected. At this point, seemingly simple changes to a system will require the modification of a larger-than-expected number of modules.

3.3 Definitions of Correction Techniques

3.3.1 Security

1. **system reconfiguration:** Two approaches are possible for this set of techniques: proactive and reactive reconfiguration. These approaches are described as follows by Wolf and colleagues [Wolf 00]:

   Proactive reconfiguration adds, removes, and replaces components and interconnections to cause a system to assume postures that achieve enterprise-wide intrusion tolerance goals, such as increased resilience to specific kinds of attacks or increased preparedness for recovery from specific kinds of failures. Proactive reconfiguration can also cause a relaxation of tolerance procedures once a threat has passed, in order to reduce costs, increase system performance, or even restore previously excised data and functionality. In a complementary fashion, reactive reconfiguration adds, removes, and replaces components and interconnections to restore the integrity of a system in bounded time once an intrusion has been detected and the system is known or suspected to have been compromised. Recovery strategies made possible by reactive reconfiguration include restoring the system to some previously consistent state, adapting the system to some alternative non-compromised configuration, or gracefully shedding non-trustworthy data and functionality. In our view, proactive and reactive reconfiguration are two sides of the same coin that can be profitably unified into a coherent and comprehensive survivability mechanism.

2. **shutdown components:** This technique consists of shutting off a component when it is identified as compromised.
3. **disable compromised access points:** In this technique, an access point that is identified as compromised is disabled, but the subsystem in which it was located is not removed from the system.
4. **restore components:** In this technique, components are returned to the system for use when they are considered safe and intruder free.

3.3.2 Performance

5. **load balancing:** This technique consists of judiciously and transparently redistributing the load of the system among its nodes to achieve the maximum overall performance. These algorithms attempt to equalize the loads on all computers involved [Shivaratri 92].

6. **service degradation/interruption:** In this technique, a low-priority or non-critical service is selected for degradation/interruption. In this way, a higher priority service can take advantage of the freed resources.

3.3.3 Dependability

7. **damage confinement:** This technique attempts to constrain the spread of errors from one part of the system to another and to simplify damage assessment and error recovery [Taylor 99].

8. **backward recovery:** This technique replaces an erroneous state with some previous state known to be free of errors (e.g., via checkpoints or recovery blocks).

9. **forward recovery:** This technique repairs the system state by finding a new one from which the system can continue operation. Exception handling is one method of forward recovery.

10. **compensation:** This technique uses redundancy to mask an error and allow transformation (perhaps via reconfiguration) to an error-free state. Compensation is achieved by modular redundancy. Independent computations are voted on and a final result is selected. Majority voting might be supplemented with other algorithms to mask complex, Byzantine faults. Modular redundancy requires independence among component failures. This is a reasonable assumption for physical faults but a questionable one for software design faults (e.g., N-version programming).

3.3.4 Modifiability

11. **refactoring:** This technique is defined by Fowler as "... the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure" [Fowler 99].

12. **reengineering:** This technique is the examination and alteration of a subject system to reconstitute and implement it in a new form [Chikofsky 90]. For our purposes, reengineering includes both software and hardware.

13. **wrapping:** This technique consists of surrounding legacy systems with a software layer that hides the unwanted complexity of the old system and exports a modern interface. Wrapping removes mismatches between the interface that is exported by a software artifact and the interfaces that are required by current integration practices [Comella 00].

4 Results and Further Work

Sections 4.1 through 4.3 show the results of this research. These results are presented in the form of three matrices, one for each group of techniques: promotion, detection, and correction.

These matrices are very easy to read. For each technique, there are rows and columns based on the quality attribute that it promotes. On the far right of each row of each summary matrix is a number that corresponds to the detailed matrices provided in the appendices of this report. These detailed matrices contain the explanations of all the interactions presented in the row.

As a convention, in all the matrices presented in this report, if a cell or row is blank (i.e., its color is grey), the interaction is not explained due to the symmetry of the matrix. Because the research conducted assumed that the interactions are symmetric, all the matrices are upper triangular.

Following each summary matrix is a subsection dedicated to the analysis of the overall interaction of the quality attributes, with each other and with the other three attributes, in terms of the techniques that were surveyed. For each quality group interaction, a score is given. This is a simple way to determine how well a group of techniques interacts with itself or other groups. To calculate this score, the following rules were followed:

1. If the interaction was positive, one point was added (+1).
2. If the interaction was negative, one point was subtracted (-1).
3. If the interaction was neutral, the score wasn't modified (0).
4. If the interaction could not be determined—signified by a question mark (?)—a tolerance coefficient (+/- 1) was added. These cases were added together and were not included with the value calculated from Rules 1, 2, or 3 above.

4.1 Promotion Matrix Summary

				Security					Performance						Dependability			Modifiability			
				Promotion																	
				Cryptography	Access control	Survivability	Threat assessment	Vulnerability analysis	RMA	Performance engineering	Data replication	Process replication	Data division	Concurrency (Process division)	Testing	Markov Modeling	Replication	Change scenarios	Separation of concerns	Information hiding	Detailed matrix
Security	Promotion	Cryptography	1				=			=		=	=			=	=	=			1
		Access control	2						=	=			?			=		=			2
		Survivability	3																?		3
		Threat assessment	4						=	=									?		4
		Vulnerability analysis	5						=	=		=					=				5
Performance		RMA	6													=				=	6
		Performance engineering	7													=			?		7
		Data replication	8										=						=		8
		Process replication	9										=					=	=	=	9
		Data division	10											?	=						10
		Concurrency (process division)	11																?	?	11
Dependability		Testing	12															=			12
		Markov modeling	13																		13
		Replication	14																		14
Modifiability		Change scenarios	15																		15
		Separation of concerns	16																		16
		Information hiding	17																		

Analysis

Within security (9)

Different techniques that promote security can be combined with positive results in most cases. All other interactions are neutral. Therefore, all of these techniques can be combined without reducing security.

Within performance (-8 +/- 1)

Although the interaction between performance techniques is varied, the great majority of those interactions is negative. This would suggest that usually only one performance technique should be used for any given system.

Within dependability (1)

The only interaction that could result in reduced dependability is that of testing and replication. All other combinations of techniques can be used without a negative effect on dependability.

Within modifiability (3)

The techniques examined for this quality always result in a positive interaction. Therefore, they can be combined freely without affecting modifiability.

Security and Performance (-13 +/- 1)

Overall, security and performance techniques don't seem to interact positively. Most of the interactions are negative, followed by neutral interactions. Very few (3 out of 24) are positive.

Security and Dependability (-3)

These techniques have varied interactions. No trend is apparent for them, so an architect must be careful when trying to promote these two qualities simultaneously.

Security and Modifiability (11 +/- 2)

The large majority of the interactions among techniques for these two qualities is positive. Two of the interactions, related to separation of concerns, depend on the particular case that is being studied. Overall, however, modifiability techniques can be applied without reducing the system's security and vice versa.

Performance and Dependability (-3)

There is no clear trend for the interactions among the techniques from these two qualities. An architect should be very careful when using those techniques.

Performance and Modifiability (8 +/- 3)

Overall, the interactions are always positive or neutral with the sole exception of the interaction between process division and change scenarios.

Dependability and Modifiability (8)

The techniques have a clear positive interaction between them. Therefore, a dependable system is likely to be modifiable, and a modifiable system is likely to accept dependability easily.

Special Cases

- Performance techniques like data replication and process division (concurrency) seem to have a negative effect in most interactions. All interactions with process division are negative or uncertain (indicated by a question mark [?]), and only 5 out of 16 interactions with data replication (about 30%) are positive.
- All three modifiability techniques (change scenarios, separation of concerns, and information hiding) have a good to very good interaction with all other techniques.
- Separation of concerns seems to be the modifiability technique whose interactions vary from system to system. It participates in four interactions that are undefined (indicated by a question mark [?]) unless a concrete system is evaluated.

4.2 Detection Matrix Summary

				Security			Performance		Dependability		Modifiability			
				Detection										
				Logging	Monitoring	Honey pot	Time-outs	Missed deadlines	Triple modular redundancy	Recovery blocks	Time assessment	Defect assessment	Impact assessment	Detailed matrix
Security	Detection	Logging	1				?	?	=	=	?	?	=	17
		Monitoring	2										=	18
		Honey pot	3						=	=	=	=	=	19
Performance		Time-outs	4											20
		Missed deadlines	5						?		=	=	=	21
Dependability		Triple modular redundancy	6								=	=	=	22
		Recovery block	7								?	?	?	23
Modifiability		Time assessment	8									?	=	24
		Defect assessment	9										=	25
		Impact assessment	10											

Analysis

Within security (3)

The techniques work very well with each other. There are only positive interactions between them.

Within performance (-1)

The only interaction between the two techniques is negative; therefore, the two techniques cannot be applied at the same time.

Within dependability (1)

In this group of techniques, the only possible interaction is positive; therefore, the techniques can be applied to the same system without problems. However, once one is adopted, the second one should follow easily.

Within modifiability (+/- 1)

The result of the interaction between these techniques is dependent on the interaction between time assessment and defect assessment. In addition, this interaction is relative to the system being evaluated. Therefore, we cannot make any conclusion about the interaction of modifiability techniques.

Security and Performance (4 +/- 2)

The interaction among the techniques examined for these two qualities is mostly positive. The effect of logging seems to be unknown when interacting with performance techniques. Concrete systems need to be evaluated.

Security and Dependability (2)

Monitoring has positive interaction with all the dependability techniques examined, while logging and honey pot are neutral. Therefore, an architect combining security and dependability techniques can do so freely.

Security and Modifiability (-2 +/- 2)

In this case, honey pot has the best interaction with modifiability techniques as it is neutral to them. Monitoring is clearly negative, while logging can be either negative or positive, depending on the system being analyzed. An architect should be careful when trying to promote both qualities. Overall, very little can be established about their interaction.

Performance and Dependability (3 +/- 1)

Time-outs can be used safely when trying to promote performance and dependability in an architecture. Missed deadlines is the technique that could potentially not have a positive interaction. Architects should be careful about missed deadlines.

Performance and Modifiability (-3)

In this case, missed deadlines is neutral to the modifiability technique used. On the other hand, time-outs do not interact well with modifiability techniques. Overall, these two qualities do not interact well.

Dependability and Modifiability (0 +/- 3)

The interaction between the techniques associated with these two qualities is uncertain. Architects should be very careful when trying to promote both.

Special Cases

Honey pot is fairly innocuous when interacting with the other techniques. The interactions are positive (indicated by a plus sign [+]) or neutral (indicated by an equal sign [=]), and there are no undefined interactions (indicated by a question mark [?]). This result was expected because honey pot lies outside the boundary of the system it protects.

4.3 Correction Matrix Summary

				Security: Correction: System reconfiguration	Security: Correction: Shutdown components	Security: Correction: Disable compromised access points	Security: Correction: Restore components	Performance: Load balancing	Performance: Service degradation/interruption	Dependability: Damage confinement	Dependability: Backward recovery	Dependability: Forward recovery	Dependability: Compensation	Modifiability: Refactoring	Modifiability: Reengineering	Modifiability: Wrapping	Detailed matrix
Security	Correction	System reconfiguration	1								=	=	=	?		=	26
		Shutdown components	2					=			?			?			27
		Disable compromised access points	3				=	=			?	?	=	=	=		28
		Restore components	4					=		=	=	=	=	=	=		29
Performance		Load balancing	5						=	?	?		=	?			30
		Service degradation/ interruption	6										=	=			31
Dependability		Damage confinement	7													=	32
		Backward recovery	8											=		=	33
		Forward recovery	9											=		=	34
		Compensation	10											=		=	35
Modifiability		Refactoring	11														36
		Reengineering	12														37
		Wrapping	13														

Analysis

Within security (-1)

The interaction between techniques used to promote security is weak to bad. Only "restore components" seems to be applicable independent of the other techniques used.

Within performance (0)

Performance techniques are independent of each other, so they can be applied without risks.

Within dependability (6)

Dependability techniques are enhanced by the presence of each other. An architect can comfortably use more than one of them to improve dependability without worrying about their interactions.

Within modifiability (-3)

The modifiability techniques don't seem to interact with each other gracefully. We therefore advise that only one of them be applied for a given system.

Security and Performance (5)

Approximately half of the interactions are positive, while the other half is neutral. Therefore, these techniques can be combined freely.

Security and Dependability (1 +/- 3)

There is no concrete pattern of interaction for these two groups of techniques. Backwards recovery and the disabling of compromised access points add most of the uncertainty to this interaction; therefore, architects should keep these techniques in mind as possible sources of architectural mismatches.

Security and Modifiability (1 +/- 2)

The overall interaction between techniques for these two qualities is close to neutral. Yet, the individual interactions are spread over all possibilities. Each interaction needs to be considered in isolation.

Performance and Dependability (2 +/- 2)

Although service degradation/interruption has a positive interaction with all the dependability techniques, load balancing is dependent on the technique with which it interacts. Therefore, no general rule can be derived for these qualities.

Performance and Modifiability (0 +/- 1)

This case is similar to that of performance and dependability; no general rule can be established for them.

Dependability and Modifiability (-5)

Techniques that belong to these two qualities tend to have neutral or negative interactions. In particular, reengineering is negatively affected by all dependability techniques.

Special Cases

- Disabling compromised access points and restoring components has mostly positive or neutral interactions (except for a couple of undefined interactions (indicated by a question mark [?]) with other techniques.
- Reengineering has mostly negative interactions with other techniques.
- Except for its negative interaction with reengineering, compensation can be used with any other technique without concerns.

4.4 Further Work

This work is not finished and will probably never be. It needs to be updated and extended to cover all the techniques in use by current practitioners. Techniques that are currently leaving the research laboratories because a practical application has been found for them should also be included. We encourage readers to send us their feedback regarding improvements to this report and the usability of the summary matrices.

5 Summary of Appendices

In the following appendices, every row presented in the previous summary matrices is presented in the form of another matrix. These matrices have a detailed description of why we believe that a particular interaction is positive, negative, neutral, or cannot be determined without assessing a concrete system. Each matrix has one row per technique that belongs to the group being analyzed (promotion, detection, or correction) and is color-coded accordingly.

As mentioned at the beginning of this document, the following convention was followed when analyzing the interaction between two techniques.

	The two techniques collide, and an architect may find it very difficult to support the two techniques in the same architecture.
	The two techniques work very well with each other; they may even facilitate each other. In this case, an architect will be encouraged to use both techniques together.
=	The two techniques are independent of each other. They can coexist in the same architecture without disturbing or helping each other.
?	The type of interaction between these two techniques (positive or negative) depends on the system being studied. The result of the interaction cannot be generalized.

Grey rows correspond to interactions that were not analyzed because we assumed that the interactions were symmetric.

Appendix A – Promotion Techniques Matrices

Matrix 1 – Interactions with Cryptography

				Interactions with cryptography	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		Cryptography can be used to authenticate users of a system, thereby providing access control to it. Therefore, the two techniques have a positive interaction.
		3	Survivability		If cryptography is used for the data stored in the system, it will foster survivability because even when a system has been compromised, its data may not be. The same is true if the survivable part of the system uses cryptography to communicate between its components. It is less likely that an intruder will be able to compromise the survivable part of the system. Thus, there is a positive interaction between the two techniques.
		4	Threat assessment	=	Threat assessment may lead to the use of cryptography, but, other than this, there is no interaction between the two techniques.
		5	Vulnerability analysis		The presence or even the possiblity/impossiblity of applying cryptography methods can change the outcome of the analysis of the system's vulnerable points. Using cryptography for authentication can make a system less vulnerable to malicious users by preventing the impersonation of valid users. Using cryptography for confidentiality can prevent eavesdroppers from stealing information "in the wire." For these reasons, the interaction of the two techniques is positive.
Performance		6	Rate monotonic analysis (RMA)		Cryptography algorithms have complexity proportional to their input size. Unless the input size is bounded or the outcome of the algorithm is not considered real-time and can be preempted, the use of cryptography represents a problem for RMA. In that case, the interaction is a negative one.
		7	Performance engineering	=	Performance engineering treats cryptography as a black-box process and abstracts its complexities. Hence, the two techniques do not interact.
		8	Data replication		Either all the data are replicated, or the software must manage different groups of replicated data, some of them with encrypted data and some without. Furthermore, unless access to data is managed at the data level and not at the group level, encrypted and non-encrypted data cannot coexist in the same group. Therefore, the two techniques have a negative interaction.

Matrix 1 – Interactions with Cryptography (cont.)

				Interactions with cryptography	
				Rel	Description
Performance (cont.)		9	Process replication	=	When replicating for performance, identical copies of a process are distributed. Hence, if such a process is already using cryptography, it will not be affected by the distribution. Therefore, the two techniques are independent of each other.
		10	Data division	=	Data will be divided into logical groups, and these groups will be subject to encryption. Data are not usually divided into non-cohesive sets. For example, address information will not be divided. This allows closely related information to be encrypted together. Therefore, whether the data have been divided does not matter to cryptography.
		11	Concurrency (process division)		When using concurrency, a process is distributed between processors. In addition to the inputs and outputs to the system using cryptography, communication between the distributed components (particularly if they are physically distributed) must also use cryptography. Although the use of cryptography may not always be necessary, the analysis of when to use it is not trivial, and the system is likely to lose performance. Therefore, the interaction between these two techniques is negative.
Dependability		12	Testing		The algorithms used for encryption or the components used for this purpose must be tested thoroughly; this effort is not trivial. Because certifying these components is difficult, these two techniques have a negative interaction.
		13	Markov modeling	=	The use of cryptography can be abstracted from the model of the system unless the cryptography algorithms are being modeled; this is not usually the case. Therefore, there is no interaction between these techniques.
		14	Replication	=	The two techniques don't interact. Replication is achieved by using multiple copies of a process. Whether this process uses cryptography doesn't change the way replication is implemented.
Modifiability		15	Change scenarios	=	Cryptography may limit change scenarios (like moving from a centralized to a distributed/concurrent system). However, these kinds of changes will probably represent large efforts and radical changes to the system's architecture, which goes against the spirit of change scenarios. As a consequence, there is no interaction between cryptography and change scenarios.
		16	Separation of concerns		The algorithms and procedures used to manage the encryption process will most likely be isolated (by applying the principle of separation of concerns). So, modifying them will not be pervasive through the system. In this case, the interaction is positive.
		17	Information hiding		The use of information hiding can complement cryptography nicely. Information hiding can reduce the need for cryptography because much information will remain local and not directly accessible. Therefore, the interaction is a positive one.

Matrix 2 – Interactions with Access Control

				Interactions with access control	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		Access control enhances survivability, and survivability requires better access control for those parts of the system that need to survive an intrusion. Those components of the system that need to survive must form a subsystem where access control is stricter than the other system's subsystems. These characteristics yield a positive interaction between the two techniques.
		4	Threat assessment		The kinds of threads that the system may have to withstand determine the levels of access control. Therefore, each technique promotes the other.
		5	Vulnerability analysis		Vulnerability analysis tries to determine the points in a system where attacks are most likely to take place. It is simplified by the presence of access control because access control bounds its work. Therefore, the interaction between access control and vulnerability is positive.
Performance		6	Rate monotonic analysis (RMA)	=	Because access control is a bounded process, it should be schedulable using RMA. In most cases, access control is not a real-time process, so it falls outside the scope of RMA. The two techniques do not interact.
		7	Performance engineering	=	Access control can be treated as a black-box process that takes some time to validate a user trying to gain access to some component. Because access control can be abstracted away from the performance-engineering model, there is no interaction between the two techniques.
		8	Data replication		If replicated data are distributed, access control will also be required, increasing the complexity of the replication mechanism. For example, if data are very difficult to calculate but can be reused once they have been calculated (like data produced by radiosity algorithms) and the data storage is distributed due to the data's volume or because the data are accessed by multiple processes, those locations where the data are stored also need to be secured. Furthermore, those locations can be under the control of different administrators, complicating the situation even more. For all these reasons, there is a negative interaction between access control and data replication.
		9	Process replication		Replication is more difficult in the presence of access control. All replicas must be synchronized with respect to access-control information by active communication to enforce a fair policy. Otherwise, an attacker could, for example, probe passwords in one machine and, when about to be locked out, try another, and keep doing this until the password is found. Therefore, the two techniques have a negative interaction.

Matrix 2 – Interactions with Access Control (cont.)

				Interactions with access control	
				Rel	**Description**
Performance (cont.)		10	Data division	?	The interaction between data division and access control can be either positive or negative. It will be positive if the division allows for different access-control mechanisms or policies to be applied to the different groups of data. It can be negative if the configuration of the access control for each group of data is different and even worse if this configuration depends on different people. For these reasons, this interaction depends on the concrete system under study.
		11	Concurrency (process division)		Access control now extends not only to the computer, the system, or even the network where a system is started, but also to every other computer, system, or network where other pieces of the process are being run. Thus, the interaction between concurrency and access control is very difficult. This interaction is very similar to that for process replication (2.9) and is therefore negative.
Dependability		12	Testing		Testing a system with access control in place adds considerable work to the testing effort. It requires more configuration to represent different combinations of users; testing it, particularly for intrusion, is difficult. For these reasons, the interaction is negative.
		13	Markov modeling	=	Modeling and access control do not interact because the use of access control can be abstracted from the model. For this reason, the two techniques are independent of each other.
		14	Replication		All the replicas must be synchronized and function as a single access point. If one system is taken out of service, its access information should remain accessible. Therefore, a system that uses the two techniques is more complex than a system that uses only one of them. For these reasons, the interaction between replication and access control is negative.
Modifiability		15	Change scenarios	=	There is very little to no interaction between these two techniques. Access control might affect change scenarios either by limiting what can change due to the need to control access or by imposing access control to changes that represent opening new access points to the system. Therefore, the two techniques are independent of each other.
		16	Separation of concerns		The interaction between access control and separation of concerns generates components that are in charge of such control. This improves the components' modifiability, making this a positive interaction.
		17	Information hiding		If information hiding is applied through a system, access control could be circumscribed to those components that hide the data for which access must be controlled. Therefore, information hiding and access control have a positive interaction.

Matrix 3 – Interactions with Survivability

				Interactions with survivability	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		A system will probably require more than one level of survivability, depending on the extent of damage that a threat can cause to a system. Therefore, threat assessment will probably help achieve a higher level of survivability in the system, making the interaction between these two techniques positive.
		5	Vulnerability analysis		Knowing what needs to survive helps a designer concentrate on making certain subsystems less vulnerable than others. From the opposite perspective, vulnerability analysis helps establish what can survive an attack. Hence, the two techniques have a positive interaction.
Performance		6	Rate monotonic analysis (RMA)		As every configuration of a survivable system is a system in itself, RMA must be applied to each of those configurations. Doing so increases the complexity of the analysis, and the result of the analysis may indicate that the system is not schedulable for one or more survivable configurations. For these reasons, the interaction between RMA and survivability is negative.
		7	Performance engineering		This is the same situation as for RMA (3.6). Thus, there is a negative interaction between these two techniques.
		8	Data replication		Data that need to survive cannot be replicated with data that may or may not survive an attack. This situation makes the creation and management of replication groups more complex and maybe less efficient, and limits what can and cannot be replicated. Therefore, the interaction between these two techniques is negative.
		9	Process replication		As replication uses identical copies of processes on multiple servers, the survivability of the overall system increases because even if a server is lost completely, other servers will pick up its workload. Furthermore, if a system is survivable, it is not made more complex in the presence of multiple servers because for all purposes, all replicas are identical. Therefore, these two techniques have a positive interaction.
		10	Data division		Data division can be used to partition the data that are used by different parts of a system. Then, if one subsystem is compromised, another might not depend on the compromised subsystem's data, which will make the system more survivable. As such, the two techniques have a positive interaction.

Matrix 3 – Interactions with Survivability (cont.)

				Interactions with survivability: Rel	Interactions with survivability: Description
Performance (cont.)		11	Concurrency (process division)		Concurrency is more difficult to achieve and exploit in the presence of a survivable system because a system may be partially compromised (e.g., one or a few servers). The critical subsystems in the compromised server must survive the attack, while the servers that have not been compromised must work as if nothing has happened. In an extreme situation, a concurrent process must be able to run as if no concurrency was possible. This scenario can become a reality if enough systems are compromised. The easiest solution to this problem is not to use concurrency for those subsystems that need to survive an attack, but this defies the purpose of concurrency. For this reason, the two techniques have a negative interaction.
Dependability		12	Testing		Survivability makes the testing effort more difficult. There is a need to simulate possible attacks and to ensure that the system configuration that survives the attack is valid. Thus, these two techniques interact negatively.
		13	Markov modeling		A system that survives an attack is, for the purposes of modeling, a subset of the original system. There can be many such subsystems, as many as the number of survivability configurations. Each of those configurations must be proven valid both individually and as a subset of the subsystem that encloses it. This validation adds a large amount of effort to the modeling of a system. Thus, there is a negative interaction between survivability and Markov modeling.
		14	Replication		This case is analogous to process replication in the case of performance (3.9). Thus, there is a positive interaction between these techniques.
Modifiability		15	Change scenarios		A system that survives an attack can be considered a case of a change scenario; therefore, the two techniques complement each other yielding a positive interaction.
		16	Separation of concerns	?	There are two possibilities to consider in the interaction between survivability and separation of concerns. If the system has been partitioned into, at least, a core set of services and additional, peripheral services, it is possible that survivability will be simplified by this partitioning. If, on the other hand, survivability was not considered when the system was partitioned, survivability may be nearly impossible unless the whole system is reengineered. It is therefore impossible to determine how these two techniques interact unless a real system is evaluated.
		17	Information hiding		Survivability should foster information hiding because information needs to be hidden from the non-survivable parts of the system. Therefore, information will most likely be held in individual components, focusing the development effort on making survivable components. This makes for a positive interaction.

Matrix 4 – Interactions with Threat Assessment

				Interactions with threat assessment	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		Vulnerability analysis and threat assessment enhance each other when combined. From the point of view of threat assessment, vulnerability analysis helps architects understand possible threats to a system. From the point of view of vulnerability analysis, a system is vulnerable only in the presence of threats. Their nature can help focus the vulnerability analysis. For these reasons, the interaction is positive.
Performance		6	Rate monotonic analysis (RMA)	=	Threat assessment has no impact on RMA calculations because threat assessment identifies intrusion possibilities, while RMA is interested only in the internals of the system. Processes related to RMA are not usually related to security.
		7	Performance engineering	=	Performance engineering creates performance models of a system. These models are independent of the presence of threats. Therefore, performance engineering and threat assessment are independent.
		8	Data replication		Threat assessment identifies intrusion possibilities for a system. Given the assessment's results, data may or may not be replicated in certain subsystems or in the system at all. Hence, the type of possible intruders limits what can be cached, and the two techniques don't interact well.
		9	Process replication		As mentioned earlier, replication makes use of multiple servers, and then assessment must be performed for each one. Each server could potentially be located in a completely different environment, and even managed by different people. Furthermore, those systems' primary functions might not be those of the replicated system. For these reasons, replication and threat assessment interact in a negative way.
		10	Data division		Each of the groups into which the data are partitioned can have different threats. Although this makes threat assessment more complex, it gives the system the flexibility to secure each data group based on its needs. Because of this partitioning, the two techniques interact positively.
		11	Concurrency (process division)		This interaction is analogous to that of process replication (4.9). The interaction is negative.
Dependability		12	Testing		Threat assessment will increase the testing effort because it requires that all relevant threat scenarios be taken into account. Threat assessment, therefore, doesn't help testing, hindering the interaction of the two techniques.
		13	Markov modeling		Thread assessment can help determine what needs to be modeled as external entities to the system and their behavior. It should also help focus the modeler on security. By determining which intrusion scenarios are possible, the modeler can use this information to represent the system as it evolves when threats become real and reduce the system's operability. Therefore, this is a positive interaction.

Matrix 4 – Interactions with Threat Assessment (cont.)

				Interactions with threat assessment	
				Rel	Description
Dependability (cont.)		14	Replication		This interaction is analogous to that of process replication and threat assessment (4.9). Then, the interaction is a negative one.
Modifiability		15	Change scenarios		Threat assessment can help create a better architecture by exposing security solutions that, although fine for the original architecture, are shortsighted considering the system's evolution. Therefore, this is a positive interaction.
		16	Separation of concerns	?	Threat assessment can drive the process of splitting the system into subsystems. Splitting may or may not be good for modifiability purposes. If it helps isolate those mechanisms that are used to handle the identified threats, the interaction with separation of concerns will be positive. If, on the other hand, it forces the separation of components against their natural grouping to support attacks, the interaction with separation of concerns will be negative. For these reasons, no conclusion is possible without evaluating a concrete system.
		17	Information hiding		If the threats identified by threat assessment are mostly related to loss of data, information hiding can be improved by ensuring that data are isolated appropriately from other system components. At the same time, threat assessment should simplify protecting data since the data will probably have only one point of access, Thus, there is a positive interaction between the two techniques.

Matrix 5 – Interactions with Vulnerability Analysis

				Interactions with vulnerability analysis	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)	=	There is no interaction between vulnerability analysis and RMA because vulnerability analysis is concerned with determining which points in a system are vulnerable, while RMA is concerned with whether the system is schedulable. Therefore, the two don't interact.
		7	Performance engineering	=	The creation of performance models of a system (performance engineering) is not affected by knowing which parts of a system are vulnerable and to what. Therefore, there is no interaction between these two techniques.
		8	Data replication		Data stored in a cache can be vulnerable to attacks. If the data are stored only in memory, they can be overwritten by a maverick process. If the data are held in secondary storage, they can be rewritten and even physically removed from the system. The use of cached data will make a system more vulnerable. The interaction, therefore, is negative.
		9	Process replication	=	A vulnerability analysis on a system needs to be performed only once, for the primary copy. All the replicas used by the selected replication technique will benefit as a side effect. Consequently, the two techniques do not interact.
		10	Data division		If data are divided within a unique server, vulnerability analysis is not affected by this technique. However, if the data are divided and spread across servers (the most likely scenario), the system becomes more vulnerable than one in which the data are centralized. This situation is particularly aggravated when multiple administrators are in charge of executing countermeasures for the vulnerabilities found in the analysis. These drawbacks make the interaction between the techniques a negative one.
		11	Concurrency (process division)		Concurrency can be complex if different vulnerability analyses identify a heterogeneous set of vulnerabilities for different servers. This will require different solutions for different servers, making the overall system more complex and less cohesive. (This assumes that not all the threats can be considered for all components due to their nature or their high cost to implement.) Therefore, concurrency and vulnerability analysis hinder each other.
Dependability		12	Testing		The testing effort should increase because it must accommodate testing for the vulnerable points that were identified. However, vulnerability analysis leads to more predictable testing because there is a target to test. Therefore, the two techniques have a positive interaction.
		13	Markov modeling		Vulnerability analysis can be used to feed a Markov model with possible failures that will trigger the dependability mechanism. Therefore, the two techniques have a positive interaction.

Matrix 5 – Interactions with Vulnerability Analysis (cont.)

				Interactions with vulnerability analysis	
				Rel	Description
Dependability (cont.)		14	Replication	=	This interaction is equivalent to the case of replication for performance (5.9). This is no interaction between the two techniques.
Modifiability		15	Change scenarios		Vulnerability analysis helps determine which change scenarios are possible given the context of the identified vulnerabilities. This is a positive interaction. Another positive interaction is presented by looking at change scenarios considering which new vulnerabilities they would either introduce or remove from the system.
		16	Separation of concerns		Vulnerability analysis helps partition a system. At a minimum, the most critical components to vulnerability should be isolated. This will help with validation and verification. As such, this is a positive interaction.
		17	Information hiding		This interaction is the same as that for separation of concerns and vulnerability analysis (5.16). Therefore, it is a positive interaction.

Matrix 6 – Interactions with Rate Monotonic Analysis (RMA)

				Interactions with RMA	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		RMA is concerned with the ability to schedule processes, and performance engineering is concerned with how long a process will take to execute; they complement each other, as one can feed information to the other. This complementary relationship is especially true regarding process synchronization, which is a particularly difficult element to analyze. The two techniques, which provide different perspectives on the problem, should help each other. For these reasons, the two techniques have a positive interaction.
		8	Data replication		RMA will not be applicable to those processes that use data replicated across systems or servers. If those processes are part of the scheduling problem that is being modeled, the two techniques have a negative interaction.
		9	Process replication		If task completion depends on distributed tasks completing their work, the task cannot be included in RMA; due to latencies, the time a task will take to complete is not bounded and therefore cannot be fed into a RMA. The two techniques, then, don't work well together.
		10	Data division		Unless the groups of data are collocated with the processes that access them, access to remote data becomes unbounded and RMA is not applicable. For this reason, the interaction is negative.
		11	Concurrency (process division)		If task completion depends on distributed tasks completing their work, the task cannot be included in RMA. Due to latencies, the time a task will take to complete is not bounded. Therefore, there is a negative interaction between the two techniques.
Dependability		12	Testing		Once RMA is done (if based on accurate data), it should reduce system testing because the system is known to be schedulable. Therefore, there is a positive interaction between the two techniques.
		13	Markov modeling	=	RMA is a modeling technique in itself, but is concerned only with performance. Therefore, it complements any other models of a system that may exist. The two techniques are independent of each other.
		14	Replication		As mentioned earlier for performance, when distributed algorithms are used for consistency checking, RMA cannot be used because the operations are not time-bounded. In this case, the interaction between these two techniques is negative.

Matrix 6 – Interactions with Rate Monotonic Analysis (RMA) (cont.)

				Interactions with RMA	
				Rel	Description
Modifiability		15	Change scenarios		Although RMA must be done for each individual change scenario, it will help understand whether the scenarios are realistic in terms of the schedulability of the system. Therefore, the interaction between these two techniques is positive.
		16	Separation of concerns		Simpler tasks are easier to analyze with RMA than complex tasks that perform many functions and are therefore not cohesive. Separation of concerns tries to make the analysis simpler too, so both techniques should foster each other.
		17	Information hiding	=	Although information hiding adds overhead to any implementation, this overhead can be factored out of RMA. Therefore, the two techniques don't interact.

Matrix 7 – Interactions with Performance Engineering

				Interactions with performance engineering	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		The presence of data replication increases the complexity of the performance-engineering model if the replication must be accurate. Although worst-case scenarios can be used, they are most likely not satisfactory (which is why the performance analysis is conducted in the first place). Therefore, performance analysis and data replication interact negatively.
		9	Process replication		Replication makes performance models more complex and requires additional work. Latencies must be identified and taken into account, particularly when considering synchronizations between processes. Doing so can make the model intractable. Therefore, there is a negative interaction between the two techniques.
		10	Data division		This is analogous to the interaction between RMA and data replication (6.8). Unless data are collocated, performance engineering becomes so complex that it might be incorrect or even intractable. There is a negative interaction between the two techniques.
		11	Concurrency (process division)		The interaction between concurrency and performance modeling has the same consequences as those of the interaction between replication and performance engineering (7.9). The two techniques interact negatively with each other.
Dependability		12	Testing		Although verification is needed for performance assumptions, if they are verified, performance and system testing should not only be easier but much more predictable than without the use of performance-engineering techniques. Therefore, there is a positive interaction between the two techniques.
		13	Markov modeling	=	As in the case for RMA (6.13), performance engineering models a system from a complementary point of view. Hence, the techniques do not interact.
		14	Replication		The interaction between replication to increase dependability and performance modeling has the same consequences as those of the interaction between replication for performance and performance engineering (7.9). The two techniques interact negatively with each other.
Modifiability		15	Change scenarios		Although each change scenario can potentially require a separate performance model, those models will help architects better understand the impact of each change scenario, and determine if the scenarios aren't plausible given the performance constraints of the system. In this case, there is a positive interaction between these two techniques.

Matrix 7 – Interactions with Performance Engineering (cont.)

				Interactions with performance engineering	
				Rel	Description
Modifiability (cont.)		16	Separation of concerns	?	This interaction depends mainly on the way in which the system was partitioned. If the main goal is to ease performance modeling, one technique is promoting the other. If the main goal is not performance related, performance analysis could be very complex. A concrete system is needed to determine how these two techniques interact.
		17	Information hiding		If correctly used, the performance model should be easier to create with information hiding because there is only one way to access/change information in the system. Once each operation is modeled, the modeler knows that this information is valid for every process/component in the system. Therefore, the two techniques have a positive interaction with each other.

Matrix 8 – Interactions with Data Replication

				Interactions with data replication	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		The use of data and process replication will allow replicated processes to access data locally and not necessarily across servers. This creates a positive interaction between the techniques.
		10	Data division	=	Once data replication is in place, the number of groups into which it is divided is not a drawback. Therefore, the two techniques are independent of each other.
		11	Concurrency (process division)		The combination of concurrency and data replication makes the resulting system very complex because replicated data need to be globally consistent. This complexity causes a negative interaction between the two techniques.
Dependability		12	Testing		Certifying that the replicated data remain consistent in the presence of server and process failures is a very complex testing activity. Therefore, the two techniques exhibit a negative interaction.
		13	Markov modeling		Data replication is accommodated in Markov models. Therefore, the two techniques have a positive interaction.
		14	Replication		The use of data replication for performance and process replication for dependability complement each other. Data replication adds to the dependability of the system, while process replication allows for processes and data to be collocated as described under process replication (8.9). Therefore, the interaction between these two techniques is positive.
Modifiability		15	Change scenarios		Change scenarios can validate whether the architecture of a system is using replicated data in a way that will prove useful when encountering foreseeable changes. Therefore, these two techniques have a positive interaction.
		16	Separation of concerns	=	Separation of concerns is concerned with functional decomposition, whereas data replication is concerned with data location. Therefore, the two techniques do not interact.
		17	Information hiding		By combining information hiding and replicated data, specialized replication policies can be used depending on the data being replicated. Information hiding also means that there is only one point of access to the data; therefore, there are no spurious accesses to data without going through the data replication interface. For these reasons, the interaction between information hiding and data replication is a desired one.

Matrix 9 – Interactions with Process Replication

				Interactions with process replication	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division	=	Process and data replication are orthogonal to each other. Because replicated processes are exact copies of each other, the problem of replicated data is solved once for the whole system of replicas. Therefore, the two techniques do not interact.
		11	Concurrency (process division)		Systems that support both concurrency and replication are very difficult to construct, test, and validate. Therefore, the interaction between these two techniques is negative.
Dependability		12	Testing		In addition to testing in isolation the process to be replicated, replication requires testing several copies of the process running at the same time. Furthermore, the subsystem that implements the replication strategy must also be tested. Much testing is required to ensure that all the replicas produce the same results and don't interact with each other in unexpected ways that could corrupt their shared data. Process replication adds considerable effort and complexity to testing. As a result, the interaction between the two techniques is negative.
		13	Markov modeling		The process replication is part of the model. Therefore, the two techniques have a positive interaction.
		14	Replication		The interaction of replication for both performance and dependability increases the number of failures that the system can sustain (albeit with degraded performance). The two techniques work in favor of each other, making this a positive interaction.
Modifiability		15	Change scenarios	=	Assuming that there is no communication between the copies and that the copies are identical, changes should need to be applied to only one of the copies. Therefore, process replication doesn't add any complexity to the change scenarios, and the two techniques do not interact.
		16	Separation of concerns	=	There is no interaction between separation of concerns and process replication because process replication is applied to complete systems, while separation of concerns is applied to individual components.
		17	Information hiding	=	This interaction is analogous to the interaction of separation of concerns with process replication (9.16); therefore, the two techniques do not interact.

Matrix 10 – Interactions with Data Division

				Interactions with data division	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)	?	The presence of data division will probably imply that the data are located on different servers. If the divided processes are collocated with the data they use, performance should be increased with respect to a monolithic system. On the other hand, if data and processes are not always collocated, performance will suffer and can potentially be worse than for a monolithic system. Therefore, a concrete system is required to determine how these two techniques interact.
Dependability		12	Testing	=	Although the configuration of a system with data division can be more complex than one without it, the testing effort is, for the most part, independent of the division. Therefore, the two techniques don't interact.
		13	Markov modeling		As divided data become a point of failure, the Markov model becomes more complex. Therefore, the two techniques have a negative interaction.
		14	Replication		Data division requires a replication scheme of its own that adds to the complexity of system replication for dependability. Therefore, the two techniques have a negative interaction.
Modifiability		15	Change scenarios		Change scenarios should help validate if a given data division is valid, not just for the current incarnation of a system, but for subsequent ones when the change scenarios are applied. Therefore, there is a positive interaction between these two techniques.
		16	Separation of concerns		Separation of concerns can be used to split data into cohesive groups (e.g., employee information from customer information). This should help create data groups that can be used by different applications and rarely cross referenced. This separation would improve performance, so the two techniques have a positive interaction.
		17	Information hiding		Information hiding allows data division to be transparent to users of the data. Therefore, the two techniques have a positive interaction.

Matrix 11 – Interactions with Process Division

				Interactions with process division	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		Testing concurrent systems is very difficult; therefore, the two techniques interact in a negative way.
		13	Markov modeling		Modeling concurrent systems is very difficult; they make the Markov model more difficult to construct accurately. Therefore, the interaction between these two techniques is negative.
		14	Replication		This interaction is the same as that for process division and process replication for performance (9.11). The interaction between the two techniques is therefore negative.
Modifiability		15	Change scenarios		Implementing change scenarios in a concurrent system is difficult. Describing them need not be difficult, but evaluating them in a concurrent system is much harder than in a non-concurrent system. Therefore, the two techniques have a negative interaction.
		16	Separation of concerns	?	A concrete system must be examined to determine whether the interaction between separation of concerns and concurrency is positive or negative. The interaction depends on what criteria were used to separate the system into components. If the components are inherently and computationally independent, concurrency can be achieved relatively easy. But if they are not independent, the useful concurrency of a system can be limited.
		17	Information hiding	?	This is a similar case to that mentioned above for separation of concerns. If the hidden data are local to where the process is executing, concurrency will not be affected and might even be fostered. If the information is not local, calls across processes are required. This scenario can limit or cancel any advantage that concurrency might have tried to achieve. Therefore, the interaction between concurrency and information hiding cannot be assessed unless a concrete system is evaluated.

Matrix 12 – Interactions with Testing

				Interactions with testing	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		
		13	Markov modeling		A good model of a system will not only help testing, it will also direct it. A model can be used to determine error conditions to test what would otherwise not be noticed. Testing and modeling interact well together.
		14	Replication		A replicated system is more complex to test because many scenarios, quite a few of which require precise timing, are required. This complexity makes the interaction between the two techniques a negative one.
Modifiability		15	Change scenarios	=	Change scenarios and testing do not interact because change scenarios are concerned with the potential of change, not change that has taken place. Although the two techniques do not interact, change scenarios, if properly documented and analyzed, can help create test plans if the change scenarios become real.
		16	Separation of concerns		Separation of concerns makes more cohesive components, which helps to simplify testing those components because test engineers need to know less to do their work. Consequently, the tests will also be simpler since components tend to be independent of each other. Therefore, these two techniques interact positively.
		17	Information hiding		The interaction between information hiding and testing is analogous to the one between separation of concerns and testing (12.16). The interaction between the two techniques is therefore positive.

Matrix 13 – Interactions with Markov Modeling

				Interactions with Markov modeling	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		
		13	Markov modeling		
		14	Replication		Markov models are created to study dependability in the presence of replication, so the two techniques have a positive interaction.
Modifiability		15	Change scenarios		The model of a system can be used to discuss change scenarios, and their impact can be established on a more solid foundation than by using intuition. For these reasons, the techniques have a positive interaction.
		16	Separation of concerns		The only circumstance when a model of a system is manageable is when separation of concerns is used to make the components cohesive enough to be tractable. Then, in most cases, one technique benefits from the presence of the other, rendering a positive interaction.
		17	Information hiding		This case is analogous to the interaction of separation of concerns and modeling (13.16). As such, the two techniques have a positive interaction.

Matrix 14 – Interactions with Replication

				Interactions with replication	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		
		13	Markov modeling		
		14	Replication		
Modifiability		15	Change scenarios		Change scenarios should help validate that the replication used in a system is valid for the evolution of that system. It can highlight what can and cannot be replicated due to the performance and survivability of the system. Therefore, the two techniques have a positive interaction.
		16	Separation of concerns		Separation of concerns helps replication because when it is applied to a system, the system's components will be more cohesive and thus easier to replicate. Therefore, the two techniques interact in a positive way.
		17	Information hiding		Information hiding helps replication because access to data is controlled from a single location, which should simplify replication. On the other hand, if commonly accessed data are not collocated in the same information-hiding module, once replication is applied to a system, many communications across processes that might not be collocated can occur, affecting performance. This problem is not a concern of dependability, so the two techniques interact in a positive way.

Matrix 15 – Interactions with Change Scenarios

				Interactions with change scenarios	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		
		13	Markov modeling		
		14	Replication		
Modifiability		15	Change scenarios		
		16	Separation of concerns		Change scenarios help identify the most convenient way to partition the system so that the possible changes are as simple as possible. Therefore, the interaction between the two techniques is a positive one.
		17	Information hiding		Change scenarios help identify what information must be hidden to simplify making those possible changes. Therefore, the interaction is positive.

Matrix 16 – Interactions with Separation of Concerns

				Interactions with separation of concerns	
				Rel	Description
Security	Promotion	1	Cryptography		
		2	Access control		
		3	Survivability		
		4	Threat assessment		
		5	Vulnerability analysis		
Performance		6	Rate monotonic analysis (RMA)		
		7	Performance engineering		
		8	Data replication		
		9	Process replication		
		10	Data division		
		11	Concurrency (process division)		
Dependability		12	Testing		
		13	Markov modeling		
		14	Replication		
Modifiability		15	Change scenarios		
		16	Separation of concerns		
		17	Information hiding		Information hiding and separation of concerns usually complement each other. Making components cohesive by hiding some particular knowledge or information, achieves separation of concerns and vice versa. The interaction between the two techniques is a desired interaction that should be sought.

Appendix B – Detection Techniques Matrices

Matrix 17 – Interactions with Logging

				Interactions with logging	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		Although monitoring can be performed without logging, it is not realistic to do so. Logging adds history and review capabilities to the system. If an intruder penetrates a system, even if the penetration is confined and doesn't cause any harm, it is difficult to discover what has happened and fix the defect that led to the intrusion without logging. Therefore, these two techniques have a positive interaction.
		3	Honey pot		This interaction is very similar to that with monitoring (17.2). Without logging, the honey pot can divert intruders, but their activities cannot be assessed. Therefore, the two techniques have a positive interaction.
Performance		4	Time-outs	?	There are two possible interactions here. In the first one, the interaction between logging and time-outs is negative because logging is the cause of the time-outs. The second one, positive, uses logging to record time-outs. If the logged time-out information contains such data as the state of the system when it timed-out, logging can help architects understand why the time-out happened in the first place. Given the preceding explanation, a concrete system must be studied to determine the interaction between these two techniques.
		5	Missed deadlines	?	This interaction is very similar to that with time-outs (17.4). If logging is used within real-time tasks, tasks can miss their deadlines. If logging is a preemptable, low-priority process, it might not generate missed deadlines. The interaction depends on how and where logging is implemented in a concrete system.
Dependability		6	Triple modular redundancy (TMR)	=	Logging is not involved in the decision-making algorithm for TMR. Therefore, these techniques do not interact.
		7	Recovery blocks	=	Logging and recovery blocks are not related. Unless security logging is a critical function for a system, recovery blocks will provide dependability for other modules.
Modifiability		8	Time assessment	?	Two possibilities arise for the interaction of logging and time assessment. On one hand, modifying a module that interacts with the logging mechanism is seldom trivial. It can lead to less accurate time assessments for the modifications. On the other hand, these concerns are relevant only if the change involves the logging mechanism. Therefore, the interaction between these two techniques can be evaluated only for concrete cases.
		9	Defect assessment	?	Logging can signal a defect and be used to narrow down defect possibilities, but only if the component that has the defect uses the logging mechanism in meaningful ways. Therefore, as was the case for time assessment (17.8), the interaction between these two techniques cannot be assessed without evaluating a concrete case.

Matrix 17 – Interactions with Logging (cont.)

				Interactions with logging	
				Rel	Description
Modifiability (cont.)		10	Impact assessment	=	Logging can be used as a proxy of the number of components affected by a defect if they use the logging mechanism and the defect is reflected in the log. However, in most cases, there is no interaction between impact assessment and logging because the majority of defects are not recorded by the logging mechanism.

Matrix 18 – Interactions with Monitoring

				Interactions with monitoring	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		The main purpose of a honey pot is to redirect an intruder to a known system to analyze the intruder. Then monitoring must be used to trigger the logging mechanism and inform operators of the intrusion. Therefore, these two techniques have a positive interaction.
Performance		4	Time-outs		Monitoring time-outs could be used to detect denial-of-service attacks and other similar forms of service degradation due to intrusions. Therefore, there is a positive interaction between the two techniques.
		5	Missed deadlines		Monitoring can also be used to detect missed deadlines as an indication of denial of service or some other form of intrusion. As in the interaction between time-outs and monitoring, these techniques have a positive interaction.
Dependability		6	Triple modular redundancy (TMR)		Monitoring can be used to detect intrusions and the replacement of good software modules with "bad" ones. If monitoring finds these fake modules, it can instruct the TMR not to use that module in the voting process. Because of this possibility, the two techniques have a positive interaction.
		7	Recovery blocks		Similar to TMR (18.6), monitoring can be used to instruct the recovery-block mechanism to consider a compromised module as failed and move on to another trusted one. Therefore, these two techniques have a positive interaction.
Modifiability		8	Time assessment		Monitoring mechanisms are usually intrusive to the components they monitor, making modifications to those components more complicated than expected and resulting in wrong time assessments. Therefore, the two techniques have a negative interaction.
		9	Defect assessment		The interaction between defect assessment and monitoring is analogous to the interaction of time assessment and monitoring (18.8). Therefore, there is a negative interaction between defect assessment and monitoring.
		10	Impact assessment	=	Unless the modification affects the monitoring system, modifications to components that are subject to monitoring are the same as those that aren't. Therefore, in most circumstances, the two techniques do not interact.

Matrix 19 – Interactions with Honey Pot

				Interactions with honey pot	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		A honey pot contains an intruder in a safe and isolated system; thus, it helps prevent time-outs due to an intrusion. Otherwise, the honey pot does not make the time-out detection algorithm simpler or more complex. Therefore, there is a positive interaction between the two techniques.
		5	Missed deadlines		A honey pot contains an intruder in a safe and isolated system. Therefore, a honey pot helps prevent missed deadlines due to an intrusion. Otherwise, the honey pot does not make the algorithm that detects missed deadlines simpler or more complex. This is analogous to the previous interaction (19.4), making the interaction a positive one.
Dependability		6	Triple modular redundancy (TMR)	=	Honey pots are independent of TMR because the TMR is implemented inside a system. In contrast, a honey pot is by definition independent from the system that implements TMR. Therefore, the two techniques do not interact.
		7	Recovery blocks	=	Recovery blocks are independent from honey pots, as was the case for TMR (19.6). Detecting a failure in a module is not affected by the presence or absence of a honey pot (which lies outside the system with the recovery block). Therefore, the two techniques do not interact.
Modifiability		8	Time assessment	=	The honey pot is independent from changes to the system. Its evolution is not tied to the system that it guards. Therefore, the two techniques do not interact.
		9	Defect assessment	=	There is no interaction between defect assessment and a honey pot for the same reasons explained above for time assessment (19.8). Then, the two techniques are independent of each other.
		10	Impact assessment	=	There is no interaction between impact assessment and a honey pot for the same reasons explained above for time assessment (19.8). There is no interaction between these two techniques.

Matrix 20 – Interactions with Time-Outs

				Interactions with time-outs	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		Time-outs and missed deadlines represent different approaches to detecting performance problems. Time-outs are concerned with subsystems or processes being alive, while missed deadlines are concerned with processes taking too long to complete. It is very difficult to combine these two techniques because a process can miss its deadline but still be alive. Conversely a process that is not alive, by definition, will miss its deadline. Therefore, these two techniques have a negative interaction.
Dependability		6	Triple modular redundancy (TMR)		Time-outs find that a subsystem or process failed to respond on time. TMR can act on this knowledge if it considers that the system is therefore not available. This relationship implies that the techniques interact in a positive way.
		7	Recovery blocks		Recovery blocks can use time-outs as a test to determine if a process should be replaced by its recovery block. Therefore, the two techniques have a positive interaction.
Modifiability		8	Time assessment		The components of the system that depend on the time-out mechanism are usually critical. This fact, combined with the use of the time-out mechanism, increases the complexity of assessing the time that modifications will require. Therefore, the interaction between these two techniques is negative.
		9	Defect assessment		Time-out mechanisms are hard to implement and change. Therefore, changes related to the time-out mechanism are likely to be error prone. As in the previous case (20.8), the techniques have a negative interaction.
		10	Impact assessment		Adding even simple functionality that must be monitored by the time-out mechanism will require more effort than adding functionality that doesn't. Therefore, the two techniques interact negatively.

Matrix 21 – Interactions with Missed Deadlines

				Interactions with missed deadlines	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		
Dependability		6	Triple modular redundancy(TMR)	?	For hard, real-time systems, a missed deadline means a value that is not valid. Missed deadlines can be used with TMR to determine which values produced for the voting process are useful. Otherwise, for non-real-time systems, the techniques don't interact. The interaction depends on the concrete system studied.
		7	Recovery blocks		Missed deadlines can be used by a recovery-block mechanism to detect that a block is no longer satisfying its mission. That block can be replaced by another block that takes less time to complete the task being monitored by the recovery-block mechanism. Therefore, the two techniques have a positive interaction.
Modifiability		8	Time assessment	=	Missed deadlines are independent of maintenance work because they are usually implemented by an independent monitoring system. Although it can be argued that modifying the monitoring system is usually difficult and the time required to make such modifications is hard to assess, we believe that they rarely occur. Therefore, the techniques do not interact with each other.
		9	Defect assessment	=	This interaction is analogous to the interaction with time assessment (21.8). The two techniques are independent of each other.
		10	Impact assessment	=	This interaction is analogous to the interactions with time assessment (21.8) and defect assessment (21.9). Therefore, there is no interaction between impact assessment and missed deadlines.

Matrix 22 – Interactions with Triple Modular Redundancy (TMR)

				Interactions with triple modular redundancy	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		
Dependability		6	Triple modular redundancy (TMR)		
		7	Recovery blocks		TMR and recovery blocks can be combined. TMR can be used as the source of decisions for recovery blocks. Therefore, the two techniques have a positive interaction.
Modifiability		8	Time assessment	=	Unless the changes are related to TMR, there is no interaction between the two techniques. When the changes are related to TMR, given its usual complexity, they will probably take longer than normal changes. However, such changes are unlikely. No interaction has been found between the two techniques.
		9	Defect assessment	=	This interaction is similar to the previous case. TMR does not imply that changes to the system will introduce many defects. This is not true, of course, if the changes are made to the TMR component. However, such changes are unlikely. Therefore, the two techniques are independent of each other.
		10	Impact assessment	=	This interaction is similar to the previous case. TMR does not imply that small changes to the system will take longer than expected. This is not true, of course, if the changes are made to the TMR component. However, these changes are unlikely. For our purposes, there is no interaction between the two techniques.

Matrix 23 – Interactions with Recovery Blocks

				Interactions with recovery blocks	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		
Dependability		6	Triple modular redundancy (TMR)		
		7	Recovery blocks		
Modifiability		8	Time assessment	?	If the recovery block controller is affected, the defect should be simple to correct due to its use of simple algorithms for the test. Then, time assessment of the defect should be accurate. On the other hand, fixing one or more of the blocks in the recovery mechanism can be very difficult to perform and estimate due to the difficulty of determining if flaws in one block are also present in others. Therefore, the interaction of these two techniques can be evaluated only for concrete cases.
		9	Defect assessment	?	The interaction between defect assessment and recovery blocks can be evaluated only for concrete systems. The reasoning for this is analogous to that for time assessment (23.8). The interaction will depend on the system under study.
		10	Impact assessment	?	The interaction between impact assessment and recovery blocks can be evaluated only for concrete systems. The reasoning for this is analogous to that for time assessment (23.8). The interaction will depend on the system being studied.

Matrix 24 – Interactions with Time Assessment

				Interactions with time assessment	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		
Dependability		6	Triple modular redundancy (TMR)		
		7	Recovery blocks		
Modifiability		8	Time assessment		
		9	Defect assessment	?	This is one of the few cases where the interaction between two techniques is not symmetrical. Time assessment does not necessarily depend on defect assessment because modifications might not be due to defects. If they are, defect assessment becomes critical for a correct time assessment. There is no interaction between the two techniques from the point of view of defect assessment. Although a defect can take a long time to correct because of its complexity, the defect's complexity is not necessarily the only reason for a long correction time. A trivial but pervasive defect can also take a long time to correct. Therefore, the interaction between the techniques can be positive or negative, depending on the situation.
		10	Impact assessment	=	These two techniques are not necessarily related. Bad time assessment for the removal of a defect does not imply that the impact of the defect is going to be either large or small. It could be known that a defect is circumscribed to a particular module, yet the time estimated to fix it can be off by two orders of magnitude (in any direction). Therefore, there is no interaction between the two techniques.

Matrix 25 – Interactions with Defect Assessment

				Interactions with defect assessment	
				Rel	Description
Security	Detection	1	Logging		
		2	Monitoring		
		3	Honey pot		
Performance		4	Time-outs		
		5	Missed deadlines		
Dependability		6	Triple modular redundancy (TMR)		
		7	Recovery blocks		
Modifiability		8	Time assessment		
		9	Defect assessment		
		10	Impact assessment	=	If a badly assessed defect is encapsulated, its impact on the overall system can be minimal. A well-assessed defect can imply either a small or large impact. Therefore, there is no interaction between the two techniques.

Appendix C – Correction Techniques Matrices

Matrix 26 – Interactions with System Reconfiguration

				Interactions with system reconfiguration	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		Shutting down components limits the amount and kind of reconfiguration that the system can support. Therefore, the two techniques have a negative interaction.
		3	Disable compromised access points		This interaction between disable compromised access points and system reconfiguration is analogous to the one between system configuration and shutting down components (26.2). Therefore, the interaction is negative.
		4	Restore components		In contrast to shutting down components (26.2), restoring components augments the possibilities for system reconfiguration. Therefore, the interaction between the two techniques is positive.
Performance		5	Load balancing		Load balancing enhances the reconfiguration of a system. Load balancing can distribute work based on load and availability. Then, when a compromised system is reconfigured, the load-balancing mechanism can ignore those components that are offline. Therefore, there is a positive interaction between the two techniques.
		6	Service degradation/ interruption		It's safe to assume that a reconfigured system still supports the core business for which it was created, even if the system is slower. Degradation, then, will serve as a guideline for system reconfiguration and vice versa. Therefore, there is a positive interaction between the two techniques.
Dependability		7	Damage confinement		Damage confinement removes a service or components from service. However, there is no provision to return it back to service, thus limiting what system reconfigurations are possible as the system loses components. Furthermore, the configurations that are possible at a given point in time depend on what has happened before that point, making reconfiguration a very complex problem, Therefore, the interaction between the two techniques is a negative one.
		8	Backward recovery	=	Backward recovery is independent from system reconfiguration because the main function of backward recovery is to hide problems at a level that is lower than the level where the need for a system reconfiguration is found. The two techniques are independent of each other.
		9	Forward recovery	=	The interaction between forward recovery and system reconfiguration is analogous to that for backward recovery (26.8). Therefore, there is no interaction between the two techniques.
		10	Compensation	=	The interaction between compensation and system reconfiguration is analogous to that for backward recovery (26.8). Therefore, they are independent of each other.

Matrix 26 – Interactions with System Reconfiguration (cont.)

				Interactions with system reconfiguration	
				Rel	Description
Modifiability		11	Refactoring	?	Depending on the scope of the refactoring effort, system reconfiguration may or may not be affected. If refactoring is applied inside a component or a subsystem that is not partitioned during system reconfiguration, the two techniques will not interact. If, on the other hand, a major refactoring needs to take place, it might be limited by the ability to reconfigure the system or might hamper the system reconfiguration if not done carefully. This implies that the interaction will vary from system to system.
		12	Reengineering	■	The reengineered system will need to support at least the same level of system reconfiguration as the initial system. Not only is this support difficult to achieve in a running system, it might also be difficult to achieve in terms of eliciting the current system's reconfiguration capabilities. For these reasons, the interaction between the two techniques is negative.
		13	Wrapping	=	System reconfiguration is concerned with the topology of the system, whereas wrapping is concerned with hiding the complexity of the system's components. Therefore, the two techniques are independent of each other. However, good insight into the different possibilities for system reconfiguration can provide the wrapping effort with good information about what components should not be split and therefore wrapped.

Matrix 27 – Interactions with Component Shutdown

				Interactions with component shutdown	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		Shutting down a component prevents disabling only the component's external interface while still allowing that component to be used for other processes (like replication). Therefore, the two techniques have a negative interaction.
		4	Restore components		Restore components allows components previously shutdown to be reintegrated with the system. Therefore, these two techniques complement each other in a positive way.
Performance		5	Load balancing	=	Load balancing does not interact with components shutting down because a component that shuts down is no longer considered part of the load-balancing set. Therefore, the techniques do not interact.
		6	Service degradation/ interruption		As components are shut down, the degradation mechanism can detect that fewer resources are available and act upon the situation. This happens because the service degradation/interruption mechanism can consider the new system configuration as one where components must be subjected to larger workloads. Therefore, the two techniques have a positive interaction.
Dependability		7	Damage confinement		Damage confinement is usually implemented by component shutdown. Therefore, the interaction between the techniques is positive.
		8	Backward recovery	?	If the component that is shut down is the only additional component of a recovery-block mechanism, the mechanism will cease to be useful, rendering the subsystem that depends on it non-dependable. If, on the other hand, the component that is shut down does not belong to the recovery-block mechanism or there are more components available, the two techniques do not interact. Therefore, the nature of the interaction between component shutdown and backward recovery depends on the subsystem under study.
		9	Forward recovery		Only one component that belongs to TMR can be shut down at any given time, otherwise the forward-recovery mechanism will not have enough modules for TMR to function. This situation limits what components can be shut down and therefore makes the interaction between forward recovery and component shutdown a negative one.
		10	Compensation		Compensation masks components that are shut down. Therefore, the two techniques complement each other, resulting in a positive interaction.
Modifiability		11	Refactoring	?	The interaction between component shutdown and refactoring is analogous to that of system reconfiguration and refactoring (26.11). Therefore, the interaction will depend on the system under study.
		12	Reengineering		Reengineering becomes more complex in the presence of component shutdown because the need for the system to continue working in the absence of some components must be considered. This interaction makes understanding the system more difficult; thus, the two techniques have a negative interaction.
		13	Wrapping		For a component to be shut down, it must be self contained; then it can be wrapped easily. Therefore, there is a positive interaction between the two techniques.

Matrix 28 – Interactions with Disabling Compromised Access Points

				Interactions with disabling compromised access points	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components	=	Restoring components and disabling compromised access points have complementary functions. Restoring components brings components that were shut down back online, while disabling compromised access points never takes a component offline. Therefore, the two techniques do not interact.
Performance		5	Load balancing	=	Disabling a component's access points doesn't mean that the component is unable to process information on behalf of a process with a larger load. Therefore, the two techniques do not interact.
		6	Service degradation/ interruption		Disabling compromised access points will trigger a degradation/interruption of services. This means that the two techniques have a positive interaction with each other.
Dependability		7	Damage confinement		Damage confinement can be implemented by disabling compromised access points. Therefore, these techniques foster each other, and there is a positive interaction between them.
		8	Backward recovery	?	There is a negative interaction between backward recovery and disabling compromised access points if the disabled component is used by the backward-recovery mechanism. An example of this situation would be when the backward-recovery mechanism must restore information from a server whose access points have been disabled. However, in some cases, the two techniques might not interact because the backward-recovery mechanism is self-contained with respect to access points. Therefore, the interaction between these two techniques can be assessed only in the presence of a concrete system.
		9	Forward recovery	?	This interaction is analogous to that between backward recovery and disable compromised access points (28.8). Therefore, the interaction between these two techniques can be assessed only in the presence of a concrete system.
		10	Compensation	=	Compensation is concerned with faulty components and masking the failure of one component in a set. This technique has no relation to disabling compromised access points. Therefore, the two techniques are independent of each other.
Modifiability		11	Refactoring	=	These two techniques are independent. Refactoring is not concerned with components that are disabled at runtime.
		12	Reengineering	=	Reengineering and disabling compromised access points are independent. Reengineering must take into account that a system needs to disable compromised access points, but this is easy to identify and does not represent a large effort compared to other tasks that must be done to reengineer a system. The two techniques are independent of each other.

Matrix 28 – Interactions with Disabling Compromised Access Points (cont.)

				Interactions with disabling compromised access points	
				Rel	Description
Modifiability (cont.)		13	Wrapping		Identifying the access points that can be disabled is a good lead into finding the interfaces that a wrapping component will have to support. Therefore, the two techniques have a positive interaction.

Matrix 29 – Interactions with Restoring Components

				Interactions with restoring components	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing	=	Load balancing expects components to go offline and come back. Therefore, restoring components does not affect load balancing.
		6	Service degradation/ interruption		Degradation allows a system to provide either fewer services or the same services with slower performance. Restoring components brings components back online. This process can return a system to its original configuration and end a degradation period. Therefore, the two techniques have a positive interaction.
Dependability		7	Damage confinement	=	Damage confinement is concerned with removing components from a system, while restoring components brings them back. Although both techniques complement each other, they do not interact because damage confinement is concerned only with shutting down faulty components.
		8	Backward recovery	=	Backward recovery is used when a component fails, not when a component is brought back online. Therefore, the techniques are independent of each other.
		9	Forward recovery	=	This interaction is very similar to that of backward recovery (29.8). Forward recovery is used when a component is detected as faulty rather than when a component is brought back online. There is no interaction between these two techniques.
		10	Compensation	=	This interaction is also similar to the interaction of restoring components and backward recovery (29.8). Compensation masks failures and takes components out of the system; compensation is not affected or improved by returning components to the system. The techniques are independent of each other.
Modifiability		11	Refactoring	=	Restoring components is not affected by code refactoring. Restoring components is a runtime activity, while refactoring is concerned only with the static view of the system. The two techniques are independent of each other.
		12	Reengineering	=	The only interaction between restoring components and reengineering is that architects must consider the need to restore components when reengineering a system. Therefore, there is no interaction between these two techniques.
		13	Wrapping		A component that can be restored is a well-defined element for wrapping purposes. Therefore, restoring components should ease wrapping, and there is a positive interaction between these two techniques.

Matrix 30 – Interactions with Load Balancing

				Interactions with load balancing	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption	=	Load-balancing techniques try to make a system more responsive or make better use of the system's resources. They are not concerned with system degradation. Whether service degradation is used depends on the load in the component. Therefore, the two techniques operate at different levels of detail and do not interact with each other.
Dependability		7	Damage confinement	?	In this case, the interaction depends on which technique is applied first. If load balancing is in place, adding damage confinement will not affect it. On the other hand, if damage confinement is present, adding load balancing will allow a system to shift work from one damaged subsystem to another. For this enhancement to be realized, load balancing must monitor services, not just servers. Therefore, this interaction depends on the concrete system under study.
		8	Backward recovery	?	If dynamic load balancing is used, the load-balancing mechanism will react appropriately to a backward recovery by moving processing to other processors that have a lighter load. If static load balancing is used, the load-balancing mechanism will ignore that a block is being recovered and continue to send it processing requests. In addition, the performance of the affected component may change due to backward recovery, requiring the load-balancing system to react accordingly. The interaction between the two techniques depends on the concrete system being analyzed.
		9	Forward recovery		Forward recovery generally implies the use of a simple algorithm to calculate a safe value. In this case, this simple algorithm is more likely to require fewer computer resources and run faster. Then, a dynamic load balancer is likely to try to pick this component for use more often than components that take longer to execute although they produce the correct answer. Therefore, the interaction between the two techniques is negative.
		10	Compensation	=	These techniques do not interact because the load balancer is concerned with larger components than compensation.
Modifiability		11	Refactoring	?	If refactoring steps outside the boundaries of a process, it will affect load balancing because the rearranged processes/modules need to support the interface used by the load-balancing mechanism. Otherwise, if refactoring stays within the boundaries of a process, the two techniques don't interact. Therefore, the interaction between these two techniques depends on the concrete case being examined.

Matrix 30 – Interactions with Load Balancing (cont.)

				Interactions with load balancing	
				Rel	Description
Modifiability (cont.)		12	Reengineering		Load balancing is a requirement that must be considered when reengineering a system. Usually, load balancing is implemented by removing state from the servers. Doing this allows for switching dynamically between components on different physical machines without having any state dependencies between a process and where it is located. Load balancing thus makes reengineering the system more costly. Therefore, the interaction is negative.
		13	Wrapping		Wrapping means hiding a subsystem to conform to a new interface. Load balancing needs an interface too, so the two techniques can be combined without problems. If the original system supported load balancing, then this interface only needs to be exposed through the wrapper interface. If the system didn't support load balancing, wrapping could be a way to achieve it. Therefore, the interaction is positive.

Matrix 31 – Interactions with Service Degradation/ Interruption

				Interactions with service degradation/interruption	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		If a degradation/interruption mechanism is in place, damage confinement should be easier to implement because if a component is removed from the system, the system can still either provide degraded performance for the component that was removed or continue to provide other services despite the removed component. Therefore, the two techniques have a positive interaction.
		8	Backward recovery		The interaction between backward recovery and service degradation/interruption is analogous to that of damage confinement and service degradation/interruption (31.7). A system implementing degradation/interruption can cope with a backward-recovery operation that might take an appreciable amount of time to complete. Therefore, there is a positive interaction between the two techniques.
		9	Forward recovery		This interaction is analogous to backward recovery and degradation/interruption (31.8). Therefore, there is a positive interaction between the two techniques.
		10	Compensation	=	Compensation masks faults by using parallel computation. Since compensation does not require service degradation/interruption, the two techniques are independent of each other.
Modifiability		11	Refactoring	=	Service degradation/interruption is concerned with services at runtime, whereas refactoring is concerned with compile-time components. Therefore, there is no interaction between the two techniques.
		12	Reengineering		Reengineering a system that allows service degradation/interruption is more complex than reengineering one that doesn't because there will be subtleties in the implementation of the original system that will be difficult to replicate in the newly reengineered system. Therefore, the two techniques have a negative interaction.
		13	Wrapping		Service degradation/interruption implies that the system under consideration has well-defined components, which must be cohesive. These components are an asset to the wrapping process, and they are good candidates to be wrapped individually. Therefore, there is a positive interaction between these two techniques.

Matrix 32 – Interactions with Damage Confinement

				Interactions with damage confinement	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		Damage confinement helps backward recovery by reducing the extent of the operation to be recovered and making the mechanism faster. Thus, the interaction between these two techniques is positive.
		9	Forward recovery		This case is analogous to that of backward recovery and damage confinement (32.8). Therefore, the two techniques exhibit a positive interaction.
		10	Compensation		Damage confinement constrains the spread of an error, thereby creating components that are good candidates for compensation. Therefore, the two techniques have a positive interaction.
Modifiability		11	Refactoring		Refactoring must limit itself to be within the boundary of the subsystems defined by damage confinement. This might not always yield the best overall structure for the subsystems. Therefore, there is a negative interaction between the two techniques.
		12	Reengineering		Similar to refactoring, damage confinement limits the range of reengineering solutions for a given system. Those components that exhibit damage confinement will probably need to remain untouched. Therefore, there is a negative interaction between the two techniques.
		13	Wrapping	=	Wrapping will probably hide the details of damage confinement, because wrapping is not usually used for components that depend on larger components inside a legacy system. Therefore, the two techniques don't interact with each other.

Matrix 33 – Interactions with Backward Recovery

				Interactions with backward recovery	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		
		9	Forward recovery		The combination of forward and backward recovery creates very powerful and fast dependability mechanisms. Which one is applied first in the error-recovery process depends on the nature of the application, but the interaction is always positive.
		10	Compensation		The module that encompasses the backward recovery will exhibit compensation characteristics. Therefore, the interaction between the two techniques is a positive one.
Modifiability		11	Refactoring	=	Refactoring is independent of backward recovery because backward recovery restores a previous state, which is a very well-defined function. Refactoring, on the other hand, is concerned with how functionality is split between components.
		12	Reengineering		Reengineering becomes more complex in the presence of backward recovery because the backward-recovery mechanism, the state space from which the system can recover, and the state space to which the system will be taken are usually not trivial. Therefore, the interaction between the two techniques is negative.
		13	Wrapping	=	This interaction is analogous to the one between wrapping and damage confinement (32.13). Therefore, there is no interaction between the two techniques.

Matrix 34 – Interactions with Forward Recovery

				Interactions with forward recovery	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		
		9	Forward recovery		
		10	Compensation		This case is analogous to compensation and backward recovery (33.10). Therefore, the two techniques have a positive interaction.
Modifiability		11	Refactoring	=	This case is analogous to refactoring and backward recovery (33.11). Therefore, the two techniques are independent of each other.
		12	Reengineering		This case is analogous to reengineering and backward recovery (33.12). Therefore, the two techniques have a negative interaction.
		13	Wrapping	=	This case is analogous to wrapping and backward recovery (33.13). Therefore, the two techniques are independent of each other.

Matrix 35 – Interactions with Compensation

				Interactions with compensation	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		
		9	Forward recovery		
		10	Compensation		
Modifiability		11	Refactoring	=	Refactoring and compensation are not related. Compensation is concerned with higher level components than refactoring. Even more, compensation is concerned with a runtime behavior (masking problems), while refactoring is concerned with a compile-time behavior (making the code structure cleaner). Therefore, the two techniques are independent of each other.
		12	Reengineering		Reengineering a system that supports compensation is complex due to the difficulty of reproducing compensation behavior accurately with respect to the original system. Therefore, there is a negative interaction between the techniques.
		13	Wrapping	=	This interaction is analogous to the ones between wrapping and damage confinement (32.13), wrapping and backward recovery (33.13), and wrapping and forward recovery (34.13). Therefore, the two techniques are independent of each other.

Matrix 36 – Interactions with Refactoring

				Interactions with refactoring	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		
		9	Forward recovery		
		10	Compensation		
Modifiability		11	Refactoring		
		12	Reengineering		The two techniques are mutually exclusive. Reengineering is used when refactoring fails due to the scope of the problem to be solved. Therefore, the interaction between these two techniques is negative.
		13	Wrapping		The two techniques are mutually exclusive. Wrapping is used to hide a system behind an interface. Wrapping is done because making small changes either are no longer cost effective or won't solve problems that the system has when integrated with new technologies or other systems. Therefore, there is a negative interaction between the two techniques.

Matrix 37 – Interactions with Reengineering

				Interactions with reengineering	
				Rel	Description
Security	Correction	1	System reconfiguration		
		2	Shutdown components		
		3	Disable compromised access points		
		4	Restore components		
Performance		5	Load balancing		
		6	Service degradation/ interruption		
Dependability		7	Damage confinement		
		8	Backward recovery		
		9	Forward recovery		
		10	Compensation		
Modifiability		11	Refactoring		
		12	Reengineering		
		13	Wrapping		Wrapping is used to prevent reengineering. Wrapping is used when it's possible to hide a system that can't be improved behind an interface, but the cost of reengineering is too large. Therefore, these techniques are mutually exclusive, and their interaction is negative.

References

URLs valid as of June 2003.

[Allen 99] Allen J. et al. *State of the Practice of Intrusion Detection Techniques* (CMU/SEI-1999-TR-028, ADA 375846). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. <http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html>.

[Boehm 78] Boehm, Barry W.; Brown, John R.; Kaspar, Hand; Lipow, Myron; Macleod, Gordon J.; & Merritt, Michael J. *Characteristics of Software Quality*. New York, NY: American Elsevier, 1978.

[Boyd 96] Boyd, Mark A. "What Markov Modeling Can Do for You: An Introduction," 1-25 (Tutorial 2C). *Proceedings of the Annual Reliability and Maintainability Symposium Tutorial Notes, the International Symposium on Product Quality and Integrity (Our 42nd Year) "New Challenges, and a Changing Environment."* Las Vegas, NV, January 22-25, 1996. Banner Elk, NC: Annual Reliability and Maintainability Symposium, Scien-tech Association, 1996.

[Chikofsky 90] Chikofsky, E. J. & Cross, J. H., II. "Reverse Engineering and Design Recovery: Taxonomy." *IEEE Software 7*, 1 (January 1990): 13-17.

[Comella 00] Comella-Dorda, S.; Wallnau, K.; Seacord, R. C.; & Robert, J. *A Survey of Legacy System Modernization Approaches* (CMU/SEI-2000-TN-003, ADA 377453). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html>.

[Ellison 97] Ellison, R. J.; Fisher, D. A.; Linger, R. C.; Lipson, H. F.; Longstaff, T.; & Mead, N. R. *Survivable Network Systems: An Emerging Discipline* (CMU/SEI-97-TR-013, ADA 341963). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997. <http://www.sei.cmu.edu/publications/documents/97.reports/97tr013/97tr013abstract.html>.

[Ellison 01] Ellison, R. J. & Moore, A. P. *Architectural Refinement of the Design of Survivable Systems* (CMU/SEI-2001-TN-008, ADA 396627). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, October 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn008.html>.

[Fowler 99] Fowler, M. *Refactoring: Improving the Design of Existing Code.* Reading, MA: Addison-Wesley, 1999.

[Helal 96] Helal, Abdelsalam A.; Heddaya, Abdelsalam A.; & Bhargava, Bharat B. *Replication Techniques in Distributed Systems.* Boston, MA: Kluwer Academic Publishers, 1996.

[IEEE 90] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Standard 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology.* New York, NY: IEEE, 1990.

[Klein 93] Klein, Mark H.; Ralya, Thomas; Pollak, Bill; Obenza, Ray; & González Harbour, Michael. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Boston, MA: Kluwer Academic Publishers, 1993.

[Krsul 98a] Krsul, I. "Computer Vulnerability Analysis." Thesis proposal. West Lafayette, IN: The COAST Laboratory, Department of Computer Sciences, Purdue University, 1998. <http://ftp.cerias.purdue.edu/pub/papers/ivan-krsul /krsul9807.pdf>.

[Krsul 98b] Krsul, I. "Software Vulnerability Analysis." PhD diss. Purdue University, 1998. <http://www.acis.ufl.edu/~ivan/articles/main.pdf>.

[Laprie 92] Laprie, Jean-Claude, ed. *Dependable Computing and Fault-Tolerant Systems, Volume 5, Dependability: Basic Concepts and Terminology.* New York, NY: Springer-Verlag, 1992.

[Lassing 02] Lassing, Nico; PerOlof, Bengtsson; van Bliet, Hand; & Bosch, Jan. "Experiences with ALMA: Architecture-Level Modifiability Analysis." *Journal of Systems and Software 61*, 1 (March 1, 2002): 47-57.

[Mitra 00] Mitra, S. & McCluskey, E. J. "Word-Voter: a New Voter Design for Triple Modular Redundant Systems," 465-470. *Proceedings of the 18th IEEE VLSI Test Symposium (VTS'00)*. Montreal, Canada, April 30-May 4, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Natarajan 00] Natarajan, Bala; Gokhale, Andy; Schmidt, Douglas C.; & Yajnik, Shalini. "DOORS: Towards High-Performance Fault-Tolerant CORBA," 39-48. *Proceedings of the 2nd International Symposium on Distributed Objects and Applications (DOA '00)*. Antwerp, Belgium, September 21-23, 2000. Los Alamitos, CA: IEEE, 2000.

[Nguyen 98] Nguyen, D. & Liu, D. "Recovery Blocks In Real-Time Distributed Systems," 149-154. *Proceedings of the Annual Reliability and Maintainability Symposium. International Symposium on Product Quality and Integrity*. Anaheim, CA, January 19-22, 1998. New York, NY: IEEE, 1998.

[Powell 88] Powell, D.; Bonn, G; Seaton, D.; Verissimo, P.; & Waeselynck, F. "The Delta-4 Approach to Dependability in Open Distributed Computing Systems," 246-251. *Proceedings of the Eighteenth International Symposium on Fault-Tolerant Computing*. Tokyo, Japan, June 27-30, 1988. Washington, D.C.: IEEE Computer Society Press, 1988.

[Russell 91] Russell, Deborah & Gangemi, G T., Sr. *Computer Security Basics*. Sebastopol, CA: O'Reilly and Associates, 1991.

[Savolainen 00] Savolainen, Juha. "Improving Product Line Development with Subject-Oriented Programming." A position paper for the International Conference on Software Engineering, Workshop on Multi-Dimensional Separation of Concerns in Software Engineering. Limerick, Ireland, June 4-11, 2000. New York, NY: Association for Computing Machinery, 2000. <http://www.research.ibm.com/hyperspace/workshops/icse2000/Papers/savolainen.pdf>.

[Shivaratri 92] Shivaratri, N. G; Krueger, G P.; & Singhal, M. "Load Distributing for Locally Distributed Systems." *Computer 25,* 12 (December. 1992): 33-44.

[Smith 02] Smith, Connie U. & Williams, Lloyd G. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Solutions.* Boston, MA: Addison Wesley, 2002.

[Taylor 99] Taylor, David J. "Practical Techniques for Damage Confinement in Software," 132-143. *Proceedings of Computer Security, Dependability, and Assurance: From Needs to Solutions.* York, United Kingdom and Williamsburg, VA, July 7-9, 1998. Los Alamitos, CA: IEEE Computer Society, 1999.

[Viega 02] Viega J. & McGraw, G. *Building Secure Software: How to Avoid Security Problems the Right Way.* Boston, MA: Addison-Wesley, 2002.

[Von Neumann 56] Von Neumann, J. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," 43-98. *Automata Studies, Annals of Mathematics Studies, no. 34.* Edited by C. E. Shannon and J. McCarthy. Princeton, NJ: Princeton University Press, 1956.

[Wolf 00] Wolf, A. L.; Heimbigner, D.; Knight, J.; Devanbu, P.; Gertz, M.; & Carzaniga, A. "Bend, Don't Break: Using Reconfiguration to Achieve Survivability," 187-189. *Proceedings of the Information Survivability Workshop 2000.* Cambridge, MA, October 24-26, 2000. Piscataway, NJ: IEEE, 2000.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	14. REPORT DATE June 2003	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Interactions Among Techniques Addressing Quality Attributes		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Heman R. Eguiluz, Mario R. Barbacci		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2003-TR-003
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2003-003
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. abstract (maximum 200 words)

There is very little published work on how techniques that promote different architectural qualities interact with each other. When developing a software system, software architects need to understand the relationships among these techniques. For example, if a system is compromised, architects must consider questions such as whether it makes sense to apply damage confinement to achieve dependability, while at the same time shutting down components to promote security. To help answer such questions, this report provides matrices in which various techniques for promoting different architectural qualities are analyzed relative to each other. Four architectural qualities were analyzed: performance, security, modifiability, and dependability. The techniques that promote each one were selected and categorized as promotion, detection, or correction. For each category, matrices are presented that provide a detailed description of why a particular interaction is positive, negative, or neutral, or cannot be determined without assessing a concrete system.

14. SUBJECT TERMS architectural qualities, dependability, interactions, matrices, modifiability, performance, security, software system, techniques			15. NUMBER OF PAGES 96
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL